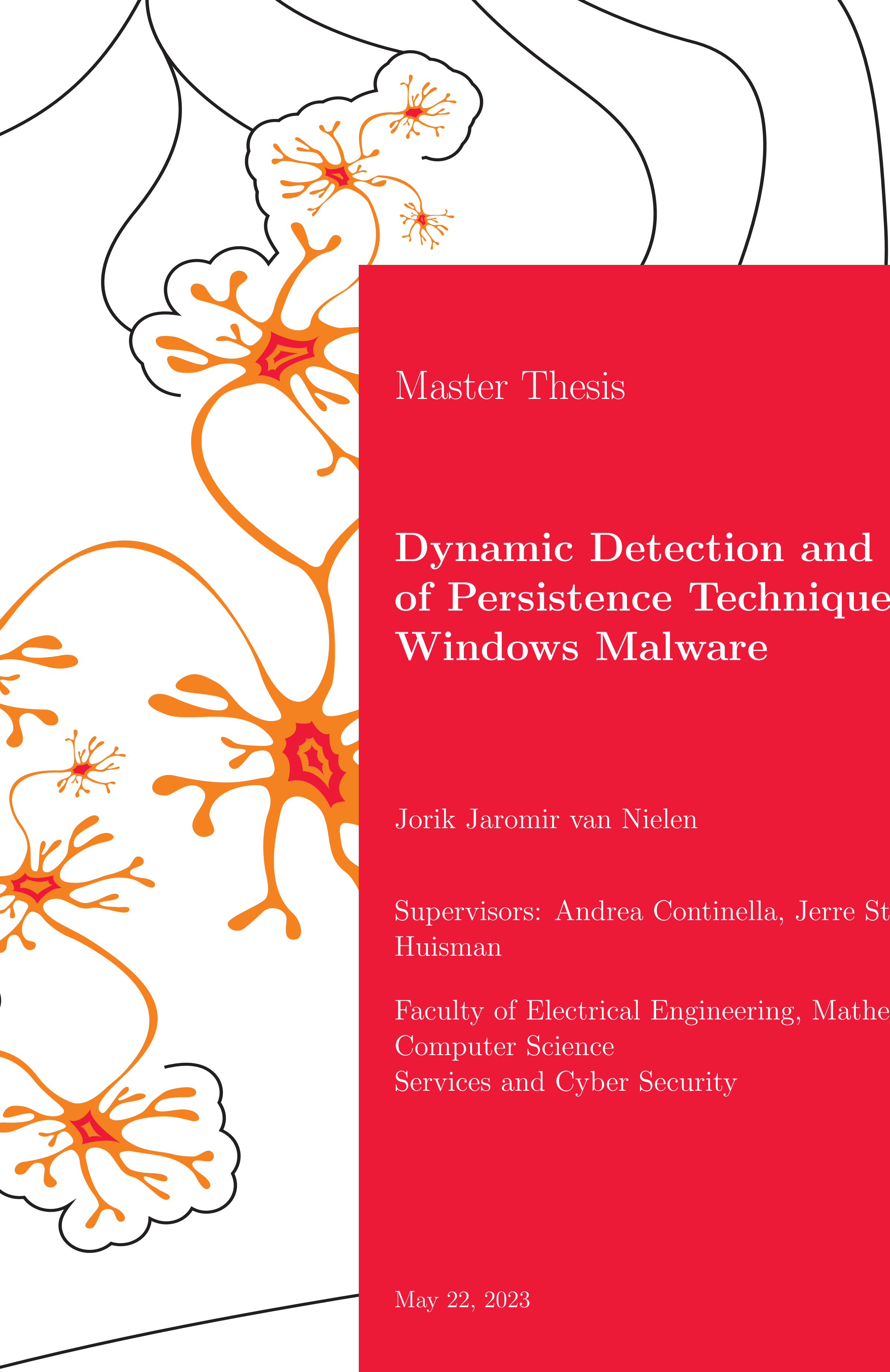


Master Thesis

Dynamic Detection and Classification of Persistence Techniques in Windows Malware

Jorik Jaromir van Nielen

Supervisors: Andrea Continella, Jerre Starink, and Marieke Huisman

Faculty of Electrical Engineering, Mathematics, and Computer Science
Services and Cyber Security

May 22, 2023

UNIVERSITY OF TWENTE.

Abstract

One of the main methods for malware to accomplish its goals is staying active on the infected machine for as long as possible. Persistence techniques are used by malware to survive reboots, user switches, and other low level events that are out of the control of the malware itself. While persistence is well known to be one of the main tactics deployed by malware, a comprehensive taxonomy on persistence techniques used by Windows malware is missing. In this paper, we provide a taxonomy of 70 distinct techniques, identify their properties, and categorize them accordingly. Additionally, we introduce a set of models to describe and detect each of the techniques. Finally, we implement a dynamic persistence detection system and analyze the adoption of persistence techniques in 5,000 real-world malware samples. We show that 16 % of the analyzed samples utilize one or multiple persistence techniques. Furthermore, we show that malware generally uses well documented techniques, but a smaller selection of samples also chooses for more exotic approaches.

Keywords: Malware, persistence, dynamic analysis, Behavior Nets, Windows

1 Introduction

Malware is an overarching term for software with malicious intent. In an age where society has become increasingly reliant on digital systems, malware has shown to be one of the main disrupters of digital machines and exploiter of their users [1]. The first known malicious sample dates back to as early as the 80s [2], and malware has been a prevalent and growing risk ever since. In 2022, 43 % of IT security decision-makers believed that their digital attack surface is spiraling out of control [3], making them susceptible to malware campaigns. Furthermore, the number of newly discovered malware samples is increasing every year, surpassing 100 million in 2021 [4]. Nowadays, criminals, activists, and nation states do not shy away from deploying malware as a means to either earn money (for example by using ransomware, such as the Wannacry malware that infected over $200,000$ machines [5]), disrupt infrastructure (such as the Mirai botnet that was able to generate more than 1Tb/s when performing DDoS attacks [6]), or steal classified information from large corporations (such as the theft of over 1TB of proprietary information of Nvidia in 2022 [7]). The impactful nature of malware, in combination with the vulnerability of organizations to malware and the large number of unique malware samples going around has made automated detection and prevention indispensable.

Antivirus software and other Endpoint Detection and Response solutions try to counteract malware both by preventing infections and detecting suspicious files and activities at runtime [8]. Detecting malware is not a trivial task however, since malware inherently tries to stay undetected for as long as possible. To effectively analyze suspicious files it is thus required to use an approach that is not easily fooled by stealth techniques. Besides the distinction between malware and goodware, the analysis and classification of malware are also relevant for incident response teams. Incident response teams are responsible for investigating malware infections and cleansing the affected machines [9]. Knowing what a malware sample has done and is doing eases the job of the incident response team and enables to act quicker on the infection. If the analysis of the malware is insufficient, the deletion attempt might be incomplete and leave (traces of) the malware on the system. By automating parts of the process of detecting well-known malicious patterns, they can spend more time analyzing newly discovered techniques [10]. Therefore, malware detection and classification is a very essential yet challenging task.

One of the main operations performed by malware is getting a foothold. This foothold, often called persistence, is needed to prevent that a simple reboot or switch of user accounts stops any further execution of the malware. Persistence techniques used by malware are vast in numbers, and as the operating system and software change new techniques are occasionally discovered [11]. While various academic and non-academic works address persistence techniques, there is limited research on a formal categorization and measurement of techniques used in real-world malware [11]–[15].

In this paper we aim to provide insight into the use of persistence techniques by malware targeting the Windows operating system. We focus on this platform specifically because more than 80 % of discovered malware samples target Windows [16], making it the most relevant for automated detection methods at scale.

1.1 Contributions

Our first contribution is a comprehensive taxonomy of Windows persistence techniques. First, we research all techniques we can find in academic works, security research blogs, online listings of persistence techniques, and persistence detection rules that have been published in various standards. We implement and verify all techniques to test if they are still functional on modern operating systems. We then study the properties of the different techniques and how they differ between individual techniques. Finally, we use the gained knowledge to categorize them based on where they reside in memory, with the goal to aid detection systems.

Our second contribution is the adaption of Behavior Nets [17] for the use of precisely describing the behavior required for all persistence techniques that we found and documented in our taxonomy. By describing the precise system calls required in a concurrent manner, the model can be used to accurately match the behavior.

Based on the taxonomy and our adaption of Behavior Nets, we present an automated detection system that can match the behavior of malware samples to the different persistence techniques. By analyzing the semantics of the behaviors, our system is able to highlight the specific persistence technique(s) that a malware sample adopts.

As a fourth and final contribution, we gauge the use of persistence techniques by malware in the wild. Leveraging the implementation of our detection system, we detect the use of persistence techniques by malware available in public databases. First, we address the difference in popularity between different techniques and different technique classes. Second, we examine in what phase of the execution of the malicious process the persistence is achieved. Last, we explore the frequency of the employment of multiple persistence techniques in a single malicious sample.

In summary, our main contributions presented in this paper are:

- **A taxonomy of persistence techniques.** We surveyed existing work on persistence techniques that function in modern versions of the Windows operating system. We present a taxonomy and a categorization based on their location in memory.
- **A model to describe persistence techniques.** We adapt Behavior Nets for persistence technique detection.
- **The design of an automated detection system for persistence techniques.** We design a system that can detect when a persistence technique is employed, based on system call data.
- **An analysis of persistence techniques adopted by malware.** We show and discuss the results of running our detection system over real-world malware.

1.2 Paper Structure

The remainder of this document is structured as follows. Section 2 provides a background on malware persistence techniques and malware detection methods. Related work and their limitations are discussed in Section 3. Next, our research on persistence techniques and the categorization can be found in Section 4. In Section 5 we describe the semantic model used to define the behavior leading to persistence. Subsequently, Section 6 features the design of our detection system that can be used to detect the different persistence techniques. In Section 7 we evaluate the performance of our system and present the results of our measurement of the adoption of persistence techniques by malware in the wild. Section 8 then discusses the impact of these results. The limitations of our work are addressed in Section 9. Finally, Section 10 draws the conclusions from our work.

2 Background

In this section we cover the nature of persistence techniques and how malicious programs leverage them. Next, we provide a background on the analysis of malware and detection techniques of malicious behavior.

2.1 Malware Persistence

The concept of persistence has been defined multiple times within the context of malware, and in essence it refers to staying active on a system for an extended period of time, despite disruptive events such as reboots [11], [12], [18]. A widely adopted definition is given by Kirillov et al., stating that persistence is “a process by which malware ensures continual execution on a system, independent of low-level system events such as shutdowns and reboots” [18]. The ATT&CK framework by MITRE defines persistence as “techniques that adversaries use to keep access to systems across restarts, changed credentials, and other interruptions” [19]. This is a broader definition that does not require any program to be executed on the targeted system to qualify as a persistence technique.

Many mechanisms can be used by malware to accomplish this resilience. Some well-known techniques to achieve persistence are adding Run registry keys [20], adding a file to the Startup folder [21], [22], and adding a Windows Service [23]–[25]. For Windows, many of these techniques have been found and documented [11]. In essence, all techniques make changes to the disk of the machine [26]. These changes persist over a reboot and results in the execution of instructions set up by the malware. A persistence technique does not necessarily launch the malware directly at boot. Some user action might be required, as long as it does not involve the user directly and willingly opening the malicious file.

Persistence techniques are largely derived from features in the Windows Operating System [11]. Instead of using the feature for their intended goal, the malware exploits the feature to become persistent. Only a few of the features that are leveraged for persistence are originally meant to ensure the execution of some code after a reboot. Instead, many of these features are meant for developers to enhance the operating system for the user by adding some extension to an existing mechanism in Windows. For example, by adding a filter to the windows search bar for a custom file extension [27]. Some persistence techniques take advantage of third party programs that are often running on the operating system. For example, a successful DLL hijack could result in running malicious operations every time the targeted third party program is started by the user [28].

For many of the objectives of malware authors, such as spying on users, mining cryptocurrencies on user machines, and turning user machines into parts of a botnet, being persistent on the system is a vital goal. To remain stealthy and undetected by virus scanners, malware authors may have incentive to get creative in their ways to persistence, instead of always using the same technique. Since virus scanners also look for suspicious files and registry entries that are related to known persistence techniques [8], it is in the advantage of malware authors to use less-known techniques. Another trick used by malware authors to fly under the radar is to make a well-known benign program persistent, and configuring the execution of this program in such a way that the malicious commands of the malware will be executed. This technique is called Living-off-the-Land [29].

2.2 Malware Analysis and Detection

To grasp how we can characterize the behavior of malware to then identify if any persistence technique is used, a good understanding of the options for malware analysis techniques is required. The two main techniques used on host machines are static analysis and dynamic analysis.

2.2.1 Static Malware Analysis

In the field of malware analysis, static analysis is defined as analyzing a potentially malicious file without running it [30]. In most cases, this entails processing the machine code and scanning it for signatures. These signatures can be strings, URLs, or malicious code sequences. Researchers who manually analyze malware samples often release these signatures to help future detection of the malware, which can then be used by for example antivirus software [8]. As malware detection has grown more effective through these means, malware authors have become smarter in the way they create their malware payloads and use various techniques to complicate the static analysis process. Already in 2006, this was a challenge dubbed the obfuscation-deobfuscation game, where the malware author tries to make it as hard as possible for the analyst [31]. Obfuscation can be categorized into different techniques [32], [33]. *Packing* is a technique that encrypts the malicious payload into an 'envelope'. When the malware is executed, the envelope is decrypted and executed. Since the initial malware file is encrypted, it is not possible to pattern-match the behavior of the malware, raising a problem for static analysis [34]. Another approach is *metamorphism*, which rearranges the machine code in a way that it looks different but is still semantically equivalent [35]. Also, registers that are used can be changed and garbage instructions - instructions that do not add to the functionality of the program - can be added. A third technique targets the matching of URL signatures by using a domain generation algorithm, which generates a domain name on request. This removes the need for storing the URL in the code as a string. It is for static analysis approaches therefore very challenging to detect if a specific persistence technique is used.

To battle obfuscation, new static analysis techniques have been developed. Control flow graph analysis creates a graph based on the binary program and uses pattern matching to look for malicious sequences. This does use more computing resources but counters various obfuscation techniques [36]. Still, many of the obfuscation techniques make static detection of persistence techniques inaccurate. Additionally, automated solutions that try to distinguish between benign and malicious programs using static analysis can unexpectedly end up differentiating between software using obfuscation techniques or not [34]. Because some benign software also uses these techniques, for example to protect intellectual property, this result is undesirable.

2.2.2 Dynamic Malware Analysis

Dynamic analysis approaches malware analysis by running the sample in a safe execution environment and monitoring its behavior [30]. The extracted behavior is then analyzed to classify the executable. Dy-

namic analysis methods do not suffer from the evasion techniques designed for static analysis, since the behavior of malware will stay mostly the same even after obfuscation techniques have been applied. In general, dynamic analysis methods consist of first capturing the behavior of the program, then abstracting the behavioral patterns, and finally classifying the sample based on the extracted patterns [37], [38].
The type of data that is collected on the process of the malware differs and can be split into three categories [39]. The *process-centric* approach keeps track of the activities of the running malware sample. In practice, Windows API calls are often recorded. This creates a very complete picture of the actions of the malware. A downside is that the amount of data that is extracted is challenging to process due to the large volume. Even more volumetric, but also more complete, is monitoring the activities of all processes active on the target machine. For example, the DRAKVUF analysis system uses this approach [40]. The *data-centric* approach does not look at the processes, but instead keeps track of the access and changes to the file system. For example, dropped files can be monitored, just as changes to relevant configuration files, and access to classified documents. While the data-centric approach only looks at system operations relating to data, it can detect many of the malicious actions of malware. However, some popular techniques including fileless malware [41] and Living-Of-The-Land malware [29] do not usually drop files, making this analysis technique less effective against them. The third and last category is the *resource-centric* approach, which relies on extracting the parameters of underlying resources. For example, it analyzes the CPU usage, the number of reads and writes to the filesystem, and the internet traffic. The resource-centric approach distinguishes itself from process-centric and data-centric approaches by not going into detail about what is happening but solely looking at the low-level data of the performance of the system. The resource-centric approach is relatively straightforward and is very effective at detecting anomalies. However, false positives are more frequent due to the lack of detail.

A second categorization that can be made relates to the way the collected data is analyzed [42]. There are *machine-learning-based* techniques that train a model on the behavior of manually labeled malware samples and use the model to classify unknown samples. The counterpart of machine-learning-based analysis techniques are *semantics-based* analysis techniques. Semantics-based analysis techniques are built on the foundation of known malicious behaviors that are performed by malicious programs. While machine learning-based detection initially results in very accurate detection and classification [43]–[45], the performance deteriorates significantly over time [46]–[48]. This drop in performance is mainly due to the rapid evolution of malware. On top of that, the initial results are sometimes also inflated because of spatial and temporal bias [49]. In other words, the training and testing datasets do not accurately match the real-world data and the time split between training and testing data is often inaccurate. To increase the performance again, the training set has to be improved, and the model has to be retrained. Semantics-based analysis techniques on the other hand require an effective model and manually written rules to identify different techniques used by malware [42]. Such rules have to be extracted from the manual analysis of malware and describe the semantics. Manual analysis is a time-costly process, but compared to machine-learning-based analysis semantics-based analysis features transparent rules can be used to accurately distinguish between different techniques used by the malware. Outdated rules and changes to existing techniques can however lead to a decrease in accuracy.

Where malware authors apply obfuscation techniques to prevent static analysis techniques from flagging their malware as malicious, analysis evasion techniques are used to limit the effectiveness of dynamic analysis. Analysis evasion targets the code-coverage limitation of dynamic analysis. The behavior of code that is not executed cannot be captured by the analysis system, limiting the completeness of the analysis results. One frequently deployed evasion technique is looking for signs in the execution environment to notice if it is being run in an analysis system [50]–[52]. If being run in an analysis system, the malicious operations are not performed. Some environments are easier to detect, such as analysis techniques that use a debugger to monitor the behavior. Recent research mostly uses VM-based analysis systems that aim to leave as few marks within the guest OS as possible [39], [43], [53], [54]. Another challenge for accurate dynamic analysis results is selecting the right execution time limit for a piece of code. Often the code coverage plateaus before two minutes, however, about 2 % of malware samples wait a set amount of time before executing any relevant code [55]. This introduces a trade-off for the analyzer between high-throughput malware analysis and accurate results. A final challenge is that the behavior of malware can differ based on the environment. The physical location, type of machine, and also time of execution all have a significant impact, making that analysis in a single sandbox will never be perfectly accurate [56].

3 Related Work

This section provides an overview of previous research in the area of persistence techniques. We will highlight the gaps that we identified and address with our research.

3.1 Persistence technique taxonomy

Several previous works have contributed towards the goal of categorizing persistence techniques. Rana et al. approached the grouping of persistence techniques by dividing them into four categories based on what part of the operating system is leveraged: Registry, Scheduled tasks, DLLs, and Services [13]. This is a great initial categorization, but requires an extension if considered for a full taxonomy, since not all persistence techniques can fit in one of these categories. For example, hijacked configuration files would fall outside all four categories. A work by Webb covers 23 techniques and groups them into five categories based on when they are executed [14]: user login, system startup, DLL injection, execution hijacking and adversary backdoors. Also, this categorization requires extension if considered for a full taxonomy, as some techniques cannot fit inside this categorization, for example the technique of hijacking a configuration file. A recent work by Villalón-Huerta, Marco-Gisber and Ripoll-Ripoll created an operating system independent taxonomy of known persistence techniques, based on the techniques featured by the ATT&CK knowledge base [11]. They divide persistence techniques into categories based on their location: Pre-OS persistence points, OS persistence points, Server-software persistence points, and User persistence points. They do not dive into the specifics of Windows persistence techniques, leaving challenges towards the goal of understanding the nature of different techniques for the use of effective detection.

Various online resources present listings of Windows persistence techniques. The ATT&CK knowledge base is a collection of real-world techniques used by adversaries and contains a wide collection of persistence techniques [19]. For each technique, the website describes the technique in detail and lists detection options. However, ATT&CK does not provide a formal classification, and the collection does lack some known techniques. A second collection is the Hexacorn blog, which lists 141 techniques with in-depth explanations [57]. The blog gives a valuable insight into the many persistence techniques. The blog posts go back to 2012, and some techniques may not be functional anymore in modern versions of Windows. The Persistence-info project features 40 persistence techniques and classifies each technique based on various criteria, such as permission required, security context, code type and more [58]. Both the Hexacorn blog and Persistence-info project leave the open challenge of classification for the use in detection.

3.2 Persistence detection

Previous work on detecting persistence techniques falls into two categories. The first is *state differential analysis.* This approach entails making a snapshot of any sorts of the system, followed by running the malware, and finishing by taking a snapshot using the same technique again [59], [60]. What the snapshot contains can differ, but often includes the file system and registry, or for our purpose the output of a persistence detection tool. A popular tool for this sort of analysis is AutoRuns [61]. AutoRuns is a high quality tool that attempts to list all active persistence on a system. The tool is closed source, and it is unclear how many techniques are detected. An alternative to Autoruns is PersistenceSniper [62]. PersistenceSniper is a PowerShell tool that can detect 43 techniques. It is actively maintained and is optimized for differential analysis of snapshot results.

The second category of persistence detection is *activity-based analysis.* Detection approaches in this category monitor the activities by the malicious sample and check if it contains any operations that lead to persistence. For example, the widely used Cuckoo Sandbox uses this approach by checking for registry operations, file write operations, and certain command calls [63]. The number of techniques covered by Cuckoo is limited however, and some techniques are not described at the lowest possible level. Besides Cuckoo, Sigma provides an open format for behavior detection and provides rules that can help to detect operations that lead to persistence [64]. Just as the Cuckoo rules, these rules are limited by their numbers and specificity.

3.2.1 Semantic models

There is a large body of knowledge on semantic models in the context of describing malicious behavior. One of the oldest and most well-known techniques is splitting the instructions executed by the programs into n-grams, which are chunks of n instructions [37], [38]. By comparing the n-grams to known malicious behavior patterns, it is possible to automatically recognize malicious patterns. However, it is trivial to circumvent this technique by using obfuscation techniques, since instructions do not necessarily have to be in the same order [32], [33].

Graph-based approaches are also well documented, and have been applied in many forms [17], [45], [65], [66]. There are two classes of graph-based semantic models. The first describes the malicious behavior in a graph form and compares this graph to the graph notation of the behavior of the malware [65], [66]. While often accurate, the downside of this type of model is the practical performance. Graph comparisons are expensive operations, especially for bigger graphs. The second class also describes the malicious behavior in a graph, but runs the operations of the malware through it as it was a Finite State Automata [17], [45], [67]. This branch of research is built on top of the well established field of process algebra [68], more specifically automata and regular expressions. A downside is that creating these graphs require manual labor to build and update.

Industry standards for describing malicious behavior are intertwined in detection rule standards. The Malware Attribute Enumeration and Characterization model (MAEC) is a language developed by MITRE and aims specifically at grouping behavior and attributes of malware [69]. MAEC uses JSON objects to represent actions on the system and has as one of its main goals to be an unambiguous language that can be used as a standard in malware description. A downside of this model is its simplicity, as one can only describe single operations and not how they relate to one another. A similar model that suffers the same limitations are Sigma detection rules [64].

3.3 Persistence in the wild

Oosthoek and Doerr took an earlier look on the use of malware techniques used by samples found in the wild [70]. They analyzed malware samples from 951 different families that were first observed between 2007 and 2018. They used a public online malware analysis tool to perform the analysis and monitored the use of 6 of the most used persistence techniques. These limitations leave some open challenges in this area of research.

3.4 Open challenges

We conclude that the following challenges remain to be addressed:

- The creation of a comprehensive and up-to-date taxonomy of functional Windows persistence techniques, including a categorization for the use of detection.
- The selection of a semantic model that can effectively describe persistence techniques for the use in detection systems.
- An analysis of the adoption of persistence techniques by recent malware samples found in the wild.

We aim to address these challenges in the remainder of this thesis.

4 Classification of Persistence Techniques

In order to detect and identify different persistence techniques used by malware, we conduct a large scale field research on known persistence techniques. The main objective is to compile a comprehensive list that includes as many techniques as possible. To achieve this goal we refer to academic works, online listings of persistence techniques, and malware-related blog posts published by security researchers. To make a proper selection, techniques are selected matching Definition 1. We chose to bring forward a new definition instead re-using one of the existing definitions as presented in Section 2.1. By adding constraints on the actions of both the attacker and the user to the definition, we simplify the discussion on what techniques should be included.

Definition 1. *A persistence technique is an operation that results in the execution of the attacker's code even after the termination of all malicious processes, without the user's direct and willful execution of the malicious code, and without any further actions by the attacker.*

Additionally, we narrow down on techniques in alignment with the following criteria:

- The technique should target the Windows operating system. Techniques for MacOS, Linux and other operating systems are out of scope.
- The technique should be reported functional in a version of Windows 10 or above.

We identified 70 distinct persistence techniques. The full description of each of the techniques can be found in Appendix B.

We created a functional, low-level reimplementation for each of the identified techniques. Our implementations rely solely on Windows API calls where possible, resulting in portability across Windows version that use different NT kernels. Only two techniques make use of additional DLLs. This concerns Windows Management Instrumentation (WMI) subscriptions [71]–[77] and Background Intelligent Transfer Service (BITS) jobs [78]–[82]. WMI subscriptions leverage a Windows features that links the execution of an executable to a Windows event, such as the user logging in. BITS jobs enable the execution of an executable when a background file transfer succeeds or fails. The use of additional DLLs is a necessity since both techniques require writes to custom databases with custom file formats. As the only way to change the persistent state of the Windows Operating System is through system calls, the reimplementations represent a ground-truth of when a persistence technique is used. To get a better understanding of the similarities and differences between persistence techniques, we defined a list of properties. The properties are inspired by the Persistence-info catalog [58] and further refined by the results of our analysis of the different techniques. The characteristics are split into two classes. The requirements class describes the conditions that the process attempting to achieve persistence must satisfy. If one of the requirements is not met, the process cannot obtain persistence. The second class lists the properties of the resulting persistence. They describe what the qualities are of the resulting persistence, in other words the properties of the persistent code and its process. Table 1 lists eight persistence techniques and their properties. The full table can be found in Appendix A. We will now discuss the various properties.

Requirements. To achieve persistence, each technique has or does not have the following requirements:

- **File System Write.** The technique writes to the file system. For example, a line has to be added to a configuration file. This property can either be true or false.
- **Registry Write.** The technique writes to the Windows Registry. For example, the malicious command has to be added to a specific key. This property can either be true or false.
- **Elevated Privileges.** The technique requires some elevated privileges. For example, administrator privileges are required to write to a specific registry key. For most techniques this property is either true or false. Some techniques work without elevated privileges, but have advantage of elevated privileges. For example, with elevated privileges the Run registry keys can be changed for all users on the system, while without elevated privileges only the Run registry keys of the current user can be changed.
- **Additional Software.** The technique requires some additional software. For example, the technique only works when Microsoft Office is installed. This property can either be true or false.
- **Interprocess communication.** The technique requires another process to perform an operation in order to work. For example, BITS jobs are stored on disk, but only the BITS service has write access to this file, thus it is required to perform the operation through this service. This property can true, false, or optional.

Resulting Persistence. The achieved persistence is different with respect to the following aspects:

- **Elevated Privileges.** The technique results in elevated privileges. For example, a technique will execute a command at boot with System privileges. This property can either be true or false.
- **Moment of Execution.** The moment that the persistence will trigger. For example, a technique will execute an executable when the user opens the File Explorer. We identified four different classes for this property. The first class is *execution at boot.* This comprises the entire boot process, including processes started after the user logs in. Besides logging in, techniques in the class do not require any user action. The second class is *execution after a user action.* This includes techniques that are activated directly by a user action. For example, a DLL hijack ensures malicious code is executed when the user opens their internet browser. The third class is *execution at a system event.* These events are not the direct results of user actions. For example, a program crash and failure to connect to a remote host are two events that would fall under this class. The fourth class is *periodic execution.* Techniques in this class ensure that the persistent code is executed at a time interval, for example every hour. Some persistence techniques can have the option to combine multiple of the classes.
- **Destructiveness.** The technique breaks a normal function of the operating system, which can be noticed by the user. For example, the screensaver does not show up or the operating system hangs. Some techniques are destructive, but with extra effort can be turned into non-destructive. An example is a DLL hijack. The DLL provided by the attacker is loaded instead of the intended DLL, which can result in a crashing program or limited functionality. However, through a technique called DLL proxying, the original functionality can be retained, while also achieving the persistence for the attacker [93].
- **Execution Type.** The type of instructions that the technique will execute. We found five different code types. First, some techniques results in the execution of a command in the Windows Command prompt or PowerShell. Second, tech-

	Requirements					Resulting persistence			
Name	**File System Write**	**Registry write**	**Elevated privileges**	**Interprocess communication**	**Additional Software**	**Elevated privileges**	**Moment of execution**	**Destructive**	**Execution type**
Databases									
Other									
BITS jobs [78]–[82]	✓	✗	✗	✓	✗	✗	P	✗	EwA
WMI subscriptions [71]–[77]	✓	✗	✓	O	✗	✓	E/P	✗	C
Registry									
Run registry keys [20]	✗	✓	✗	✗	✗	✗	B	✗	C
cmd AutoRun [83]	✗	✓	✗	✗	✗	✗	U	✗	C
Databases and Filesystem									
Registry and Operating System Files									
AppInit DLL [84]–[86]	✓	✓	✓	✗	✗	✗	U	✗	D
Screensaver [87]	✓	✓	✗	✗	✗	✗	E	✗	E
File system									
Operating System Files									
DLL hijack [28]	✓	✗	✗	✗	✗	✗	U	✗	D
lnk shortcuts [88], [89]	✓	✗	✗	✗	✗	✗	U	✗	EwA
Additional software files									
Browser extensions [90], [91]	✓	✗	✗	✗	✓	✗	U	✗	S
Windows Terminal Profile [92]	✓	✗	✗	✗	✓	✗	U	✓	C

Table 1: A subset of the persistence techniques that we identified and that meet our criteria. For the Elevated Privileges as a requirement, S = System privileges and TI = TrustedInstaller privileges. For Interprocess communication, O = optional. For the moment of executions, we have B=boot, U=user action, E=system event, P=periodically. For Code type we have C=command, E=exe, EwA=Exe with arguments, D=DLL, S=script for third party program. The full table can be found in Appendix A.

niques can result in launching an executable file. Third, the executable file is launched and there is the option to pass arguments to the program. Note that this is a significant difference from the second class, since this means that benign programs such as PowerShell can be used for malicious actions, and the placement of an executable file is not required. Fourth, code can be provided in dynamically linked libraries. Fifth and final, code can be provided as a scripting language that is run within another program. An example is JavaScript that is run inside the browser.

We classify the different persistence techniques based on where they reside in the operating system. This categorization helps with bringing more structure into the body of knowledge on Windows persistence techniques, as well as it can aid in effective detection of the different techniques. We specified the following classes:

- **File system.** Techniques in this class write files directly to the disk. We split this class into two subclasses:
    - **Operating system files.** Techniques in this class place files in folders that are used for the operating system. For example, a matching location would be the System32 folder. A technique of this class is DLL hijacking. When executed successfully, placing a DLL at a tactical location results in a regularly running process loading that DLL instead of the original DLL.
    - **Additional software files.** Techniques in this class place one or multiple files in folder that are used by additional software packages. For example, a technique can place files in folder for Microsoft Excel templates. The exact location can differ based on the installation of the software, making persistence techniques in this class harder to detect. The Windows Terminal Profile persistence technique is in this class. By changing a JSON file in the installation folder of the Windows Terminal, an executable can be set to be executed every time the terminal is started.
- **Databases.** Techniques in this class write to local databases. Technically, these databases are still local files. However, writing to these files is generally not performed through file write operations. We split this class into two subclasses:
    - **Registry.** Techniques in this class make changes to the Windows Registry. The Windows Registry is a database that is part of the Windows operating system. It stores information that is continually accessed by the operating system itself and many applications [94]. Making changes includes creating new keys, changing values and deleting entries. Windows has separate system calls to make changes to the Registry. An example of a technique that is in this class is the Run Registry keys technique [20]. By adding a key to for example HKEY_CURRENT_USER\Software \Microsoft\Windows\CurrentVersion\Run and setting the value to the path of the desired executable, one can ensure execution at boot.
    - **Other databases.** Techniques in this class make changes to local databases other than the Windows Registry. An example is the WMI database, which is used by the Windows operating system to store WMI subscriptions, i.e. actions that should be performed when certain system events occur [71]–[77]. The files that contain these databases are often opened by a continually running system process, preventing other processes to make changes directly to the file.

5 Modelling Malware Behavior

In this section, we describe how we model the behavior that leads to persistence.

To become persistent on a system, malware has to make changes to the state of the operating system. Independently of if the malware runs as an executable, through another application, or any other way, system calls are made to the operating system kernel to change its state [95]. This is the most low-level building block, so also the most stealthy malware samples have to exhibit this behavior. Since we aim to detect the persistence for any malware, we observe the behavior at this level. The collected behavior consist of a set of system calls, each containing a timestamp, system call name, and input arguments.

5.1 Requirements

To accurately describe and match the behavior that will result in persistence of the malware on a victim machine, we have implemented a model to describe the different behaviors. The requirements we set for this model to suit our needs, are as follows:

1. The model should match system calls and their input arguments. For example, if we want to match a file opening operation, we want to make sure we match the system call of opening a file, and the location that is passed in the arguments.

2. The context of operation should be taken into account. For example, when a technique requires writing the name of an executable to the registry and the placement of this file, then the model should be able to match the location placed in the registry to the file that was written to. By taking the context into account, the number of false positive matches can be limited.

3. The model should be able to distinguish between operations that can happen in any order and operations for which the order matters. For example, if a file needs to be written and a registry value has to point to this, these two operation can occur in any combination. However, for creating and then writing to a registry key, the order does matter.

4. The computational complexity of matching the behavior of malware to the different models should be linear preferably. To be able to analyze thousands of samples each consisting of thousands of operations in a timely manner, the model should support lightweight matching.

5.2 Behavior Nets

Behavior Nets [17] have been proposed and successfully used before in similar circumstances. Besides that it was used for detecting process injection instead of persistence techniques, the requirements are very similar. The model was designed specifically with system calls and their input arguments in mind, matching our first requirement. Our second requirement is also met, since the context of operations can be taken into account through the use of tokens. The third requirement of matching operation is met through the state-based model. Finally, the computational complexity with the regard to the number of system calls is fairly efficient, matching our fourth requirement. In short, a Behavior Net consists of places, partial transitions functions, and tokens. The former two are similar to states and transition functions in the more renowned Finite State Automata, respectively. Places can hold tokens, and a valid token is required for a partial transition to a next place to happen.

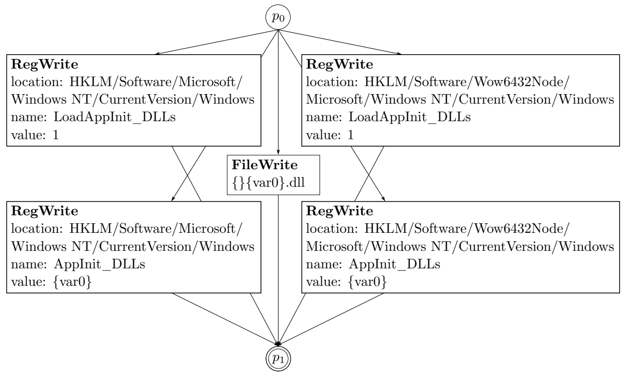

p_0
RegWrite
location: HKLM/Software/Microsoft/Windows NT/CurrentVersion/Windows
name: LoadAppInit_DLLs
value: 1
RegWrite
location: HKLM/Software/Wow6432Node/Microsoft/Windows NT/CurrentVersion/Windows
name: LoadAppInit_DLLs
value: 1
FileWrite
{}{var0}.dll
RegWrite
location: HKLM/Software/Microsoft/Windows NT/CurrentVersion/Windows
name: AppInit_DLLs
value: {var0}
RegWrite
location: HKLM/Software/Wow6432Node/Microsoft/Windows NT/CurrentVersion/Windows
name: AppInit_DLLs
value: {var0}
p_1

Figure 1: The Behavior Net that describes the detection rule for the AppInit DLLs persistence technique.

Figure 1 shows the Behavior Net that is used to detect the AppInit DLLs persistence technique. p_0 is that starting place that contains a token at the start. System calls that match the partial transition functions and their values pass tokens to the next place. Note that the occurrences of `{var0}` should match to get to the final place.

5.3 Detection rules DSL

We introduce a Domain Specific Language (DSL) that we use to write detection rules. Rules written in the DSL can be directly translated to valid Behavior Nets that can be used for detection. The aims of the DSL are threefold. The rules should be:

- Straightforward to write by hand;
- Readable, both by humans and machines;
- Capable of describing the behavior conforming the requirements set in Section 5.1.

Our DSL utilizes YAML, a serialization language which is known for being easy to write and read, for both humans and machines [96]. Our DSL is constructed of blocks, that can either be individual system calls or a composition of multiple system calls. We use three different types of compositions. Firstly, the '`AND`' composition requires all system calls to be performed. Secondly, the '`OR`' stands for an inclusive 'or' and requires only one of the system calls to be performed. Lastly, the '`SEQUENTIAL`' composition requires all system calls to be performed in the correct order. The system call notation requires the name of the system call, and optionally contains the location (for example where a file write is performed to), name (for example the name of the registry value), and the value (for example what value is written to the registry). The strings that are used to describe the location, name, and value can contain wildcards (denoted as '{}') and match values across different operation (denoted as 'var0', where the 0 can be replaced by any positive integer). An example where this last feature is useful is when the value written to the registry has to match the location of a file placed on the disk. This could result in the file write system call location value and the registry write system call value sharing the same string, namely '{var0}'. The syntax of our DSL takes on the following formal grammar form:

```
⟨detection rule⟩       ::= ⟨block⟩

⟨block⟩                ::= ⟨syscall⟩
                         | ⟨and-block⟩
                         | ⟨or-block⟩
                         | ⟨sequential-block⟩

⟨syscall⟩              ::= SYSCALL:
                           operation: ⟨operation⟩
                           (location: ⟨string⟩)?
                           (name: ⟨string⟩)?
                           (value: ⟨string⟩)?

⟨operation⟩            ::= ⟨string⟩

⟨and-block⟩            ::= AND:
                           (-⟨block⟩)+

⟨or-block⟩             ::= OR:
                           (-⟨block⟩)+

⟨seq-block⟩            ::= SEQUENTIAL:
                           (-⟨block⟩)+
```

5.3.1 Examples

Listing 1 shows the detection rule for the AppInit DLLs persistence technique [84]–[86] as an example of the use of our DSL. The base block is the `AND`-block at line 1. This `AND`-block composes two sub-blocks that must both be matched in order for the rule to return positive. The order of the two sub-blocks does not matter. The first sub-block is the `OR`-block at line 2, which features two `AND`-blocks. One of the blocks is for the 32-bit system variant, while the other is for the 64-bit variant. In both `AND`-blocks we see two `SYSCALL`-blocks. They specify the path of the Registry key, the name of the value and the actual value. In both 32-bit and 64-bit variant, the value `LoadAppInit_DLLs` is set to 1, see line 4 and 15. In the second system call, at line 9 and 20, `AppInit` is set. The `{var0}` can be seen as a regex wildcard, but must match over both system calls. The second sub-block of the base block is defined on line 25. This is again a `SYSCALL` block and specifies a file write operation. Note that the location on line 27 consists of three parts. The first part is `{}`, which is a wildcard and can take any value. The second part is `{var0}`. This value has to match the other occurrences of `{var0}`, defined at line 13 and 24. The third and final part of the location string is `.dll`, matching this exact string.

The detection rule in Listing 1 translates directly to the behavior net shown in Figure 1. The circles represent states, with p_0 being the initial state. Tokens can move to a new state in the case of a system call matching the system call in the rectangles.

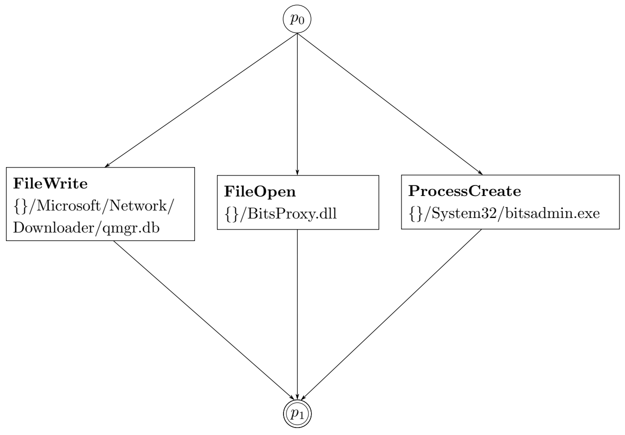
```
1  AND:
2    − OR:
3      − AND:
4        − SYSCALL:
5            operation: RegWrite
6            location: "HKLM\\Software\\
                 Microsoft\\Windows NT\\
                 CurrentVersion\\Windows"
7            name: "LoadAppInit_DLLs"
8            value: "1"
9        − SYSCALL:
10           operation: RegWrite
11           location: "HKLM\\Software\\
                 Microsoft\\Windows NT\\
                 CurrentVersion\\Windows"
12           name: "AppInit_DLLs"
13           value: "{var0}"
14     − AND:
15       − SYSCALL:
16           operation: RegWrite
17           location: "HKLM\\Software\\
                 Wow6432Node\\Microsoft\\
                 Windows NT\\
                 CurrentVersion\\Windows"
18           name: "LoadAppInit_DLLs"
19           value: "1"
20       − SYSCALL:
21           operation: RegWrite
22           location: "HKLM\\Software\\
                 Wow6432Node\\Microsoft\\
                 Windows NT\\
                 CurrentVersion\\Windows"
23           name: "AppInit_DLLs"
24           value: "{var0}"
25   − SYSCALL:
26       operation: FileWrite
27       location: "{}{var0}.dll"
```

Listing 1: Our DSL rule to describe the AppInit DLLs persistence technique

The DSL rule in Listing 2 describes the BITS jobs persistence technique [78]–[82]. There are three system operations that would trigger a positive detection outcome. The first is directly writing to the database that stores the BITS jobs, which can be found in the path described on rule 4. The second system call, starting on line 5, describes the opening of the `BitsProxy` DLL. This part of the detection rule matches when the COM DLL is used to create the BITS job. Lastly, the system call on line 8 describes the launch of the `bitsadmin` executable, which can also be used to create BITS jobs. Figure 2 shows the matching Behavior Net model.

```
1  OR:
2    − SYSCALL:
3        operation: FileWrite
4        location: ”{}\\Microsoft\\
             Network\\Downloader\\qmgr.db
             ”
5    − SYSCALL:
6        operation: FileOpen
7        location: ”{}\\BitsProxy.dll”
8    − SYSCALL:
9        operation: ProcessCreate
10       location: ”{}\\System32\\
             bitsadmin.exe”
```

Listing 2: Our DSL rule to describe the BITS jobs persistence technique



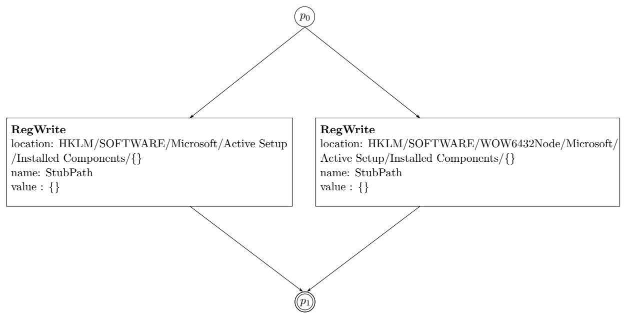
p_0
FileWrite
{}/Microsoft/Network/Downloader/qmgr.db
FileOpen
{}/BitsProxy.dll
ProcessCreate
{}/System32/bitsadmin.exe
p_1

Figure 2: The Behavior Net that describes the detection rule for the BITS jobs persistence technique.

Similarly, Listing 3 shows the rule to describe the Active Setup persistence technique [97], [98]. This is a technique from the class Registry, so only registry changes have to be detected. The Behavior Net for this rule is shown in Figure 3.

```
OR:
  − SYSCALL:
      operation: RegWrite
      location: ”HKLM\\SOFTWARE\\
          Microsoft\\Active Setup\\
          Installed Components\\{}”
      name: ”StubPath”
      value: ”{}”
  − SYSCALL:
      operation: RegWrite
      location: ”HKLM\\SOFTWARE\\
          WOW6432Node\\Microsoft\\
          Active Setup\\Installed
          Components\\{}”
      name: ”StubPath”
      value: ”{}”
```

Listing 3: Our DSL rule to describe the Active Setup persistence technique



p_0
RegWrite
location: HKLM/SOFTWARE/Microsoft/Active Setup/Installed Components/{}
name: StubPath
value : {}
RegWrite
location: HKLM/SOFTWARE/WOW6432Node/Microsoft/Active Setup/Installed Components/{}
name: StubPath
value : {}
p_1

Figure 3: The Behavior Net that describes the detection rule for the Active Setup persistence technique.

6 Detection system design

The subsequent step in our research is to design a system that can automatically detect the persistence techniques a malware samples adopts, if any. A simplified data flow diagram of our detection system is illustrated in Figure 4. It includes some details of our implementation.

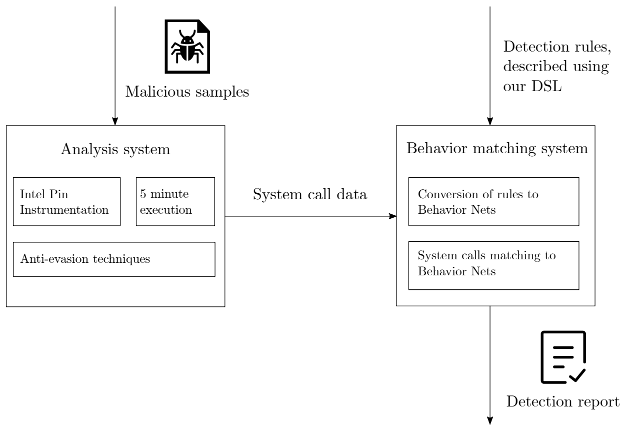

Malicious samples
Detection rules, described using our DSL
Analysis system
Intel Pin Instrumentation
5 minute execution
Anti-evasion techniques
System call data
Behavior matching system
Conversion of rules to Behavior Nets
System calls matching to Behavior Nets
Detection report

Figure 4: Simplified data flow diagram of the full persistence detection system.

Our detection system consists of two subsystems: the analysis system and the behavior matching system. The former takes a software sample as input and executes it in an analysis environment. This environment should behave as a normal Windows machine, with the addition that it logs system calls. Optimally, the output should be a timestamped data flow of system calls, including their argument. In practice, only a small selection of the system calls and their arguments are relevant for the use-case of persistence detection. The system call data is passed on the second sub-system: the behavior matching system. Alongside the system call data, the detection system takes detection rules as input. The detection rules are written in the DSL described in Section 5.3. The behavior matching system loads the detection rules into behavior nets and runs the system calls through them in the order of the time stamps. For each of the detection rules, the behavior matching system will return a positive match if the behavior net ends in an accepting state. Additionally, the behavior matching system can add context to the final detection report by logging what system calls resulted in the match.

6.1 Implementation

For our research we successfully implemented the detection system. For the analysis system we used a system from our partner university EURECOM, France. The system uses Intel PIN [99] to dynamically instrument binaries. In essence, system calls made by the program are rerouted through custom procedures and can therefore be logged [100]. The detection system was originally designed to detect and mitigate evasive techniques deployed by malware. For example, code injection techniques are redirected to instrumented processes. This makes it perfect for our use-case as well since we aim to detect persistence techniques performed by any malware. Samples were run for 5 minutes each and the relevant system calls were monitored. While the instrumentation results in a slower execution time of approximately factor two [99], the execution time should still allow for more than 98 % of samples to perform all their operations within the first two minutes [55].

We implemented the behavior matching system in Python. We chose this language to enable quick development of a functional proof of concept of the system. The system itself is split into three parts: the detection rule parser, the behavior data cleaner and the behavior matcher. Here we briefly discuss the function of each of these parts:

The detection rule parser loads the YAML files that have been written in accordance with our DSL, and builds a behavior net that matches the rule. Before any matching is done, all behavior nets are built.

The behavior data cleaner loads the system call data and cleans the data. This includes changing file system and registry paths to make sense to the behavior matcher.

The behavior matcher goes by the system calls in chronological order and applies changes in the behavior net if any. The system call responsible for causing the change in the behavior net is passed along with the token. Tokens ending in an accepting state will thus also carry the context that caused the persistence, which is carried on into the detection report.

Since we opted to use Intel PIN for the analysis system, we introduce the limitation that only system calls of instrumented binaries can be monitored. As a result, the detection of two persistence techniques can have false positives. These are the BITS jobs technique, and the WMI subscriptions technique. Both can leverage interprocess communication using COM DLLs to perform their operations. Our current detection system does not instrument the processes contacted through COM DLLs, as these processes are already running, and thus cannot log the system calls performed by these processes. While we cannot get precise matches, we can detect when these processes are used by the malicious samples. While it is likely that these processes are only called upon when they are used for persistence, it is not a certainty.

7 Evaluation

In this section we evaluate the performance of the detection system as presented in the previous section. First, we validate the performance of our system on a small test set. The goal of this small-scale validation is to verify that our detection is set up correctly and that our detection rules are correct. Second, we perform larger scale measurement, for which we run our detection system over 5,000 real-world malware samples. The main objectives of this measurement is to get a better understanding of what persistence techniques are adopted by malware, and how many malware samples use persistence techniques.

7.1 Detection system validation

To assess the performance of the detection system, we run our full persistence detection system over our proof-of-concept implementations of all techniques. We developed these implementations earlier to create our persistence technique taxonomy in Section 4. As explained earlier, these implementations use bare Windows API calls in order to be as low level as possible. We use the detection system as described in Section 6.1, with a Windows 10 32-bit operating system as the sandbox for the analysis system.

Of the tested 70 tested techniques, 66 were detected successfully. The four undetected techniques do not run properly on the system because our implementations use privilege escalation based on process injection. As the system redirects the process injection, the required privileges are not achieved and the techniques fail. Therefore, the behavior that leads to detection was not exhibited, and the persistence technique could not be detected. Real-world samples that use a privilege escalation technique that does not leverage process injection will be detected by our system. Besides this group of four, all techniques were detected successfully.

7.2 Measurement of adoption

We measure the adoption of malware persistence techniques on a malware collection of 5,000 samples. These samples have been randomly selected from the pool of VirusTotal [101] samples that were submitted in 2021. As a result, more than 90 % of the samples were first submitted to VirusTotal in 2021. Samples that are inactive and do not perform any system calls are left out of the dataset. Our selection of 5,000 samples represents 481 different malware families, while 1075 samples were not classified as any malware family. The maximum number of samples per malware family is 42. Just as in the validation experiment, we use the detection system as described in Section 6.1, with a Windows 10 32-bit operating system as the sandbox for the analysis system. Out of the 5,000 samples, 803 are flagged with using one or multiple persistence techniques. This is lower than expected, as being persistent is seen as one of the main objectives of many types of malware. The number of occurrences per technique are shown in Figure 5. The malware set adopts 15 of the 70 monitored techniques. The two most used techniques, DLL hijacks and Run registry keys, are together responsible for almost 75 % of the persistence techniques used by malware in our selection.

Figure 6 shows the number of occurrences of the different classes as defined in Section 4. It shows that techniques that use only one type of storage location, either a database or the file system, are more popular than techniques that use both. This is the result we expected, as having only one place of storage makes detection more difficult for antivirus products.

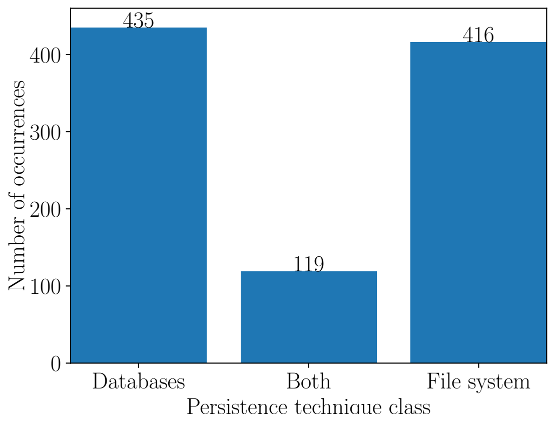


Figure 6: Detected persistence techniques occurrences per persistence class.

How many cases of persistence were detected per class of malware is shown in Figure 7. The classes of the malware samples were extracted from Virustotal reports using the AVCLass tool [102]. The first observation we make is that grayware makes up for about 40 % of our dataset, showing how popular that class is. Our second observation is that the percentage of persistent grayware samples (a bit less than 13 %) is lower than that of most other classes, that are all above 20 %, except for the Backdoor and Other class. This is expected, since grayware is in many cases expected to be executed willingly by the user. As a third observation, the adoption of persistence techniques by specifically backdoors, viruses, and worms

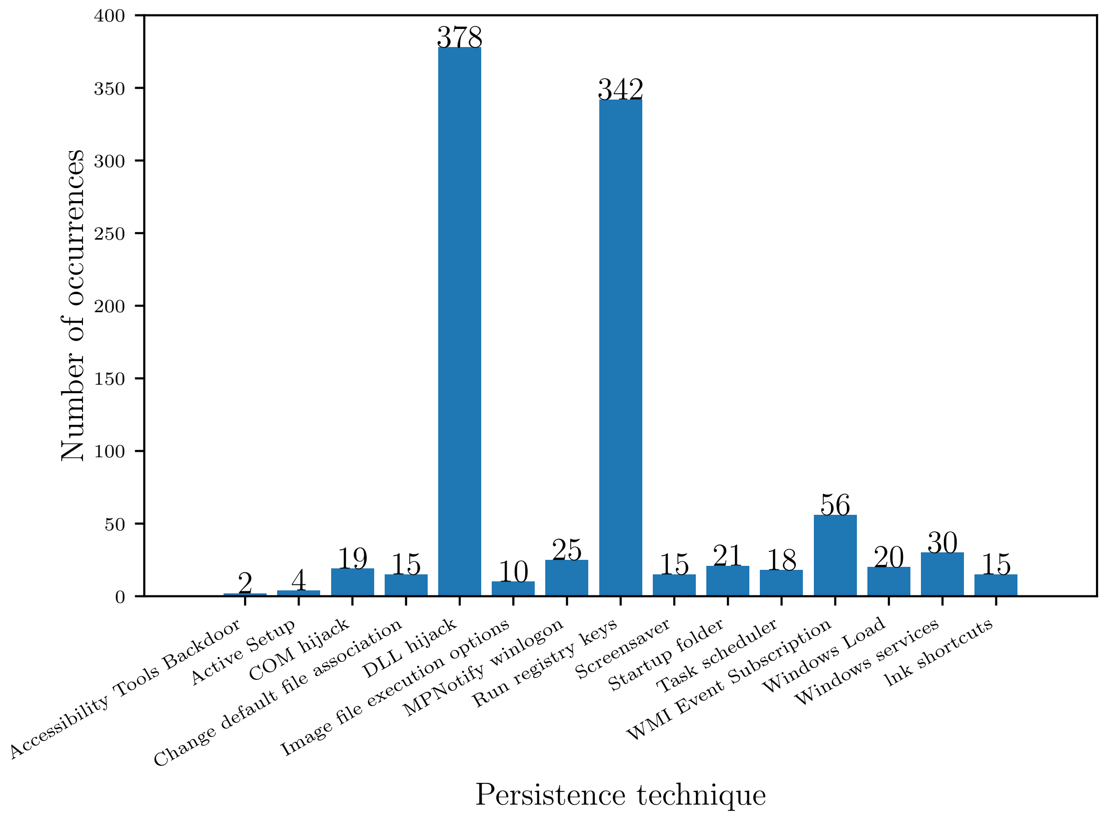


Figure 5: The number of occurrences of different persistence techniques deployed by 5000 malware samples. All techniques that were detected at least once have been included.

is lower than expected. We would expect more of the malware samples in these classes to adopt persistence because of the nature of these classes.

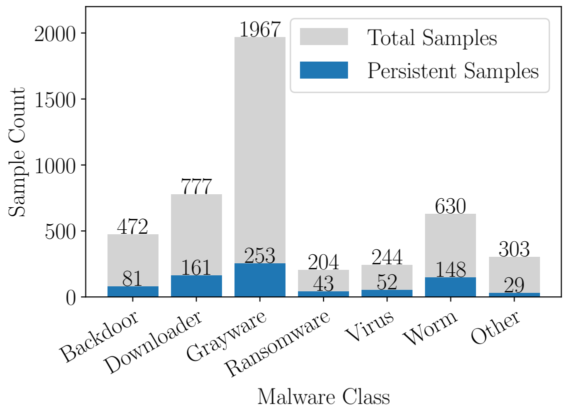


Figure 7: Detected persistence techniques occurrences per malware class. Total Samples are the total number of samples in our dataset that belong to a specific malware class. Persistent Samples are the samples of that class that we found to adopt at least one persistence technique.

Figure 8 shows when in the time of execution the persistence is achieved. For example, when a sample with an execution time of 30 seconds writes to the Run Registry key after running for 10 seconds, it would fall into the 20-30 % category. Our data shows that most malware samples either achieve persistence at the start of execution (about 13.5 % in the thirst 10 % of their execution time) or at the end of execution (approximately 14 % in the final 10 % of their execution time). However, it is important to note that the differences are not very large. For example, still more than 8 % of the samples obtain their persistence at the least popular time, which is only 6 percent point less than the most popular time.

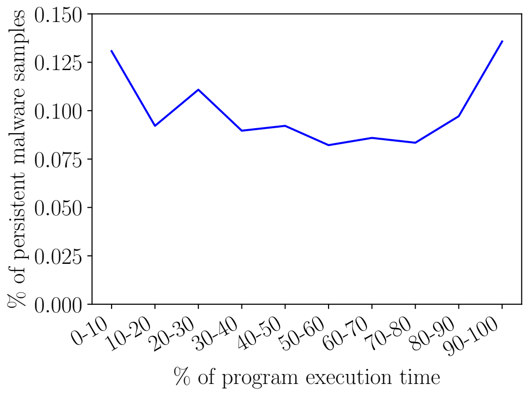


Figure 8: Observed time at which the first persistence is achieved by the malware samples, relative to the total running time of the malware.

Figure 9 shows the time in minutes it takes for the malware samples to become persistent. Note that the data has not been adjusted for instrumentation overhead time, which is approximately a doubling factor. The results are as expected, as most malicious samples perform all there malicious operations in the first two minutes of execution, or four minutes for instrumented execution [55].

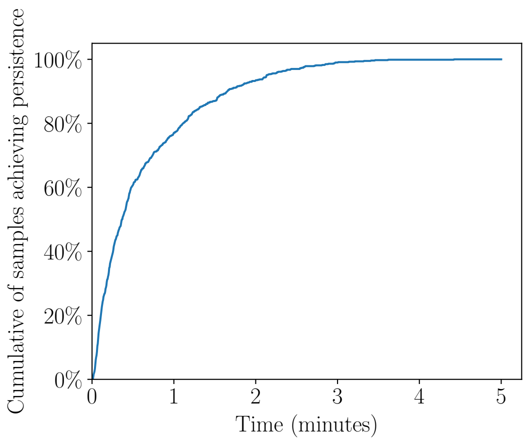


Figure 9: The cumulative percentage of malware samples that achieve their first persistence at a certain time. For example, a little less than 80 % of persistent malicious samples in our dataset require one minute of execution to achieve persistence, or half a minute in a non-analysis context.

Figure 10 shows the number of distinct persistence techniques were used per malware sample. For example, 18 samples were found to use 3 different persistence techniques. Interestingly, most malware samples rely on a single persistence technique. At the same time, a couple chose to adopt multiple, and one sample even used six.

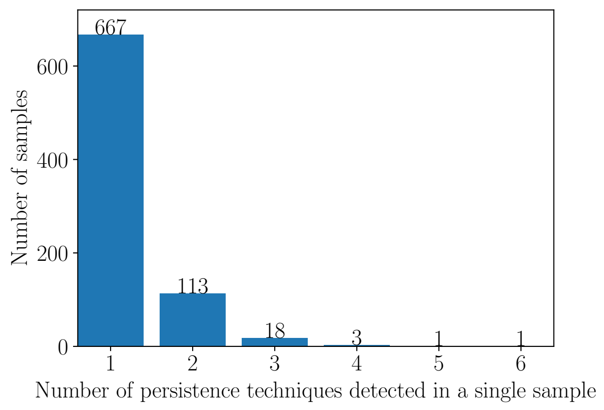


Figure 10: Observed number of persistence techniques deployed per malware sample in our dataset.

8 Discussion

In this section we discuss our findings as presented in the evaluation.

The results showed that 807 out of the 5000 analyzed malware samples exhibited behavior that was detected as persistence achieving behavior. This is a bit more than 16 % of the analyzed samples. This number seems low, and we expected more malware samples to be persistent. One apparent reason could be that this low number is related to code coverage, despite our best efforts to limit the ways that the malware can detect our detection system. To ensure the quality of our detection system, we manually performed dynamic analysis on 30 randomly selected samples for which no persistence is detected. The results showed that most did indeed not adopt any persistence technique, but two did and went undetected. For the samples that went undetected, we inspected the analysis results of our dynamic analysis system. We concluded that in some cases the analysis system does not yield the relevant system calls, but instead an unnaturally high number of system calls related to memory mapping. After further examination, we found that 1,591 of the 5,000 analysis results had this same unusual high percentage of memory mapping system calls, possibly making the results of these analyses unreliable. Additionally, as discussed in Section 2, another logical explanation for the low number is that some malware samples do not need to achieve persistence. For example, downloaders only need to download the next stage. When the dropper is unable to download the next stage, persistence is never achieved. The largest class of malware in our dataset is grayware with almost 2,000 samples (see Figure 7). This class of malware may be seen by the user as a valid program and is started by the user voluntarily. This would therefore not require any persistence. These arguments do however not explain why for example the Virus class only has a persistence technique adoption rate of about 20 %.

The distribution of what persistence technique is used (Figure 5) shows that most malware leverages two very well known techniques: Run registry keys and DLL hijacks. One reason for this could be that these techniques are functional on all versions of Windows. This means that one sample of the malware can be used on many target machines, without the need to select different techniques for different target machines.

Figure 8 shows that malware prefers to achieve persistence at the start or end of execution. For the former, the reasoning seems pretty straightforward: if the process is terminated in execution, being already persistent would ensure the survival of the malware. For the trend of gaining persistence at the end of execution, one could argue that the malware exits after the persistence has been achieved. The code that is executed by the persistence technique is then responsible for the further operations of the malware.

In general, most persistent malware samples try to leverage only one persistence technique, as was shown in Figure 10. With the knowledge that most malware samples use universally functional persistence techniques, this is not unexpected. The samples that use two or multiple techniques are often well known combinations, such as changing the default file association of a file type and adding a run registry key for a file of that type. We consider these two as separate techniques, since changing the default file association can lead to persistence when the user opens a file of that file type. However, a few of these samples also use techniques without combining them, simply increasing their persistence. Antivirus solutions that successfully remove one of the persistence mechanisms, might miss the second and remain unsuccessful in removing the malware.

For future research, it would be interesting to evaluate the results of our detection system by an automated, state differential analysis system as introduced in Section 3.2. By using a high quality snapshot tool such as Autoruns [61], the credibility of our findings could be improved. Another promising direction for future research is the automation of the discovery of new persistence techniques used by malware. Some malware samples might make use of persistence techniques that are still undocumented. By adapting the analysis system, it could be possible to detect these techniques.

9 Limitations

This section provides a view on the limitations of our research and how this may impact future work.

The first category of limitations regards the analysis limitations of our dynamic analysis approach. While there are many advantages of using dynamic analysis over static analysis as discussed in Section 2.2.2, it still has its shortcoming with regard to catching all the behavior that the malware might exhibit. Despite the anti-evasive techniques deployed in the analysis system, the malware might still conduct additional less-conventional checks before deploying its malicious operations. An example could be that it checks for the version of the operating system, and if the analysis system does not meet this requirement, we will not be able to detect the persistence techniques that it might use on operating systems that do meet the requirement. A second limitation in the category of dynamic analysis is the time limitation given to samples. As discussed in Section 6.1, the samples are run for 5 minutes, the equivalent of approximately 2.5 minutes of runtime without any instrumentation, and while this has shown to be enough for most malicious samples to exhibit all malicious behavior [55], it will not detect outliers that wait longer to gain persistence. As result, our detection results might not flag samples that require longer to become persistent.

The second category of limitations regards malware that abuses other programs to do its malicious bidding. There are various methods such as code injection [17], living-off-the-land malware [29], and regular interprocess communication. The analysis system deployed for our detection system does have mitigations to this kind of evasion as it re-routes the most used code injection techniques to instrumented binaries, but not all techniques are covered [100].

The third limitation is related to the way the system calls are monitored. Our analysis system instruments the targeted binary, more specifically it re-routes the Windows API DLL and NTDLL calls. Both of these libraries still live in user space, and to interact with the kernel they use the system call instruction. It is however possible to make a system call directly from the malicious program, circumventing the instrumentation and still accomplishing the wanted system call [103]. This is not expected to be used a lot however since these bare-bones system calls are not documented by Microsoft and can differ significantly from version to version [95].

10 Conclusion

In this paper, we presented a comprehensive taxonomy of persistence techniques targeting the Windows operating system. This is the first taxonomy of this scope, and provides a better understanding of what persistence techniques are and how they can be detected. Besides the taxonomy, we introduced a DSL to assist the writing of detection rules for persistence techniques, which can be used in our detection system. We have shown that malware actively makes use of persistence techniques, however not as much as we anticipated. While most generic malware samples deploy standard persistence techniques, more exotic samples do exist that abuse lesser known techniques. Additionally, we have shown that while most malware uses only one persistence technique, some samples do leverage multiple in an attempt increase their odds of survival.

11 Acknowledgements

We would like to thank our supervisors Andrea Continella, Jerre Starink, and Marieke Huisman. Their support and insights have been truly invaluable to our work.

References

[1] “2019 Midyear Security Roundup: Evasive Threats, Pervasive Effects,” Trend Micro. [Online]. Available: `https://documents.trendmicro.com/assets/rpt/rpt-evasive-threats-pervasive-effects.pdf`.

[2] V. Saengphaibul. “A Brief History of The Evolution of Malware — FortiGuard Labs,” Fortinet Blog. (Mar. 15, 2022), [Online]. Available: `https://www.fortinet.com/blog/threat-research/evolution-of-malware` (visited on 04/11/2023).

[3] "Defending the Expanding Attack Surface: Trend Micro 2022 Midyear Cybersecurity Report." [Online]. Available: https://documents.trendmicro.com/assets/rpt/rpt-defending-the-expanding-attack-surface-trend-micro-2022-midyear-cybersecurity-report.pdf.

[4] "Malware Statistics & Trends Report — AV-TEST." (), [Online]. Available: https://www.av-test.org/en/statistics/malware/ (visited on 06/27/2022).

[5] "What is WannaCry ransomware?" www.kaspersky.com. (Feb. 9, 2022), [Online]. Available: https://www.kaspersky.com/resource-center/threats/ransomware-wannacry (visited on 08/15/2022).

[6] "What was the Mirai botnet," Malwarebytes. (), [Online]. Available: https://www.malwarebytes.com/what-was-the-mirai-botnet (visited on 08/15/2022).

[7] M. Clark. "Nvidia says its 'proprietary information' is being leaked by hackers," The Verge. (Mar. 1, 2022), [Online]. Available: https://www.theverge.com/2022/3/1/22957212/nvidia-confirms-hack-proprietary-information-lapsus (visited on 08/15/2022).

[8] O. Sukwong, H. Kim, and J. Hoe, "Commercial Antivirus Software Effectiveness: An Empirical Study," *Computer*, vol. 44, no. 3, pp. 63–70, Mar. 2011, ISSN: 0018-9162. DOI: 10.1109/MC.2010.187. [Online]. Available: http://ieeexplore.ieee.org/document/5506074/ (visited on 04/11/2023).

[9] M. Bada, S. Creese, M. Goldsmith, C. Mitchell, and E. Phillips. "Computer Security Incident Response Teams (CSIRTs): An Overview." (2014), [Online]. Available: https://papers.ssrn.com/abstract=3659974 (visited on 04/11/2023), preprint.

[10] M. Yong Wong, M. Landen, M. Antonakakis, D. M. Blough, E. M. Redmiles, and M. Ahamad, "An Inside Look into the Practice of Malware Analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, New York, NY, USA: Association for Computing Machinery, Nov. 12, 2021, pp. 3053–3069, ISBN: 978-1-4503-8454-4. DOI: 10.1145/3460120.3484759. [Online]. Available: https://doi.org/10.1145/3460120.3484759 (visited on 05/30/2022).

[11] A. Villalón-Huerta, H. Marco-Gisbert, and I. Ripoll-Ripoll, "A Taxonomy for Threat Actors' Persistence Techniques," *Computers & Security*, vol. 121, p. 102 855, Oct. 1, 2022, ISSN: 0167-4048. DOI: 10.1016/j.cose.2022.102855. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404822002498 (visited on 04/06/2023).

[12] Z. Gittins and M. Soltys, "Malware Persistence Mechanisms," *Procedia Computer Science*, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020, vol. 176, pp. 88–97, Jan. 1, 2020, ISSN: 1877-0509. DOI: 10.1016/j.procs.2020.08.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920318342 (visited on 05/29/2022).

[13] M. U. Rana, M. Ali Shah, and O. Ellahi, "Malware Persistence and Obfuscation: An Analysis on Concealed Strategies," in *2021 26th International Conference on Automation and Computing (ICAC)*, Sep. 2021, pp. 1–6. DOI: 10.23919/ICAC50006.2021.9594197.

[14] M. S. Webb, "Evaluating Tool Based Automated Malware Analysis Through Persistence Mechanism Detection," p. 69,

[15] A. Mohanta and A. Saldanha, *Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware*. Berkeley, CA: Apress, 2020, ISBN: 978-1-4842-6192-7 978-1-4842-6193-4. DOI: 10.1007/978-1-4842-6193-4. [Online]. Available: http://link.springer.com/10.1007/978-1-4842-6193-4 (visited on 07/12/2022).

[16] "AV-TEST Security Report 2019-2020." [Online]. Available: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2019-2020.pdf (visited on 07/14/2022).

[17] J. A. L. Starink, "Analysis and automated detection of host-based code injection techniques in malware," info:eu-repo/semantics/masterThesis, University of Twente, Sep. 20, 2021. [Online]. Available: http://essay.utwente.nl/88617/ (visited on 07/13/2022).

[18] I. Kirillov and P. Chase, "Malware Attribute Enumeration and Characterization,"

[19] "MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/ (visited on 04/06/2023).

[20] “Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder, Subtechnique T1547.001 - Enterprise — MITRE ATT&CK®.” (), [Online]. Available: https://attack.mitre.org/techniques/T1547/001/ (visited on 05/21/2023).

[21] “Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder, Subtechnique T1547.001 - Enterprise — MITRE ATT&CK®.” (), [Online]. Available: https://attack.mitre.org/techniques/T1547/001/ (visited on 05/21/2023).

[22] “Startup Folder,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/startupfolder.html (visited on 05/21/2023).

[23] “Persistence with Windows Services,” PSBits. (), [Online]. Available: https://gtworek.github.io/PSBits/services.html (visited on 05/21/2023).

[24] cocomelonc. “Malware development: Persistence - part 4. Windows services. Simple C++ example.,” cocomelonc. (May 9, 2022), [Online]. Available: https://cocomelonc.github.io/tutorial/2022/05/09/malware-pers-4.html (visited on 05/21/2023).

[25] “Create or Modify System Process: Windows Service, Sub-technique T1543.003 - Enterprise — MITRE ATT&CK®.” (), [Online]. Available: https://attack.mitre.org/techniques/T1543/003/ (visited on 05/21/2023).

[26] S. Vogl, J. Pfoh, T. Kittel, and C. Eckert, “Persistent Data-only Malware: Function Hooks without Code,” in *Proceedings 2014 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2014, ISBN: 978-1-891562-35-8. DOI: 10.14722/ndss.2014.23019. [Online]. Available: https://www.ndss-symposium.org/ndss2014/programme/persistent-data-only-malware-function-hooks-without-code/ (visited on 04/11/2023).

[27] “Filter Handlers for Windows Search,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/ifilters.html (visited on 05/21/2023).

[28] W. Beukema. “Hijacking DLLs in Windows.” (Jun. 22, 2020), [Online]. Available: https://www.wietzebeukema.nl/blog/hijacking-dlls-in-windows (visited on 05/21/2023).

[29] F. Barr-Smith, X. Ugarte-Pedrero, M. Graziano, R. Spolaor, and I. Martinovic, “Survivalism: Systematic Analysis of Windows Malware Living-Off-The-Land,” in *2021 IEEE Symposium on Security and Privacy (SP)*, May 2021, pp. 1557–1574. DOI: 10.1109/SP40001.2021.00047.

[30] E. Gandotra, D. Bansal, and S. Sofat, “Malware Analysis and Classification: A Survey,” *Journal of Information Security*, vol. 2014, Feb. 20, 2014, ISSN: 2153-1242. DOI: 10.4236/jis.2014.52006. [Online]. Available: http://www.scirp.org/journal/PaperInformation.aspx?PaperID=44440 (visited on 04/11/2023).

[31] M. Christodorescu and S. Jha, “Static Analysis of Executables to Detect Malicious Patterns:” Defense Technical Information Center, Fort Belvoir, VA, Jan. 1, 2006. DOI: 10.21236/ADA449067. [Online]. Available: http://www.dtic.mil/docs/citations/ADA449067 (visited on 05/25/2022).

[32] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, “Dynamic Malware Analysis in the Modern Era—A State of the Art Survey,” *ACM Computing Surveys*, vol. 52, no. 5, pp. 1–48, Sep. 30, 2020, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3329786. [Online]. Available: https://dl.acm.org/doi/10.1145/3329786 (visited on 05/17/2022).

[33] I. You and K. Yim, “Malware Obfuscation Techniques: A Brief Survey,” in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*, Nov. 2010, pp. 297–300. DOI: 10.1109/BWCCA.2010.85.

[34] H. Aghakhani, F. Gritti, F. Mecca, *et al.*, “When Malware is Packin’ Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features,” in *Proceedings 2020 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2020, ISBN: 978-1-891562-61-7. DOI: 10.14722/ndss.2020.24310. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24310.pdf (visited on 06/09/2022).

[35] J.-M. Borello and L. Mé, "Code obfuscation techniques for metamorphic viruses," p. 10,

[36] S. Alam, I. Traore, and I. Sogukpinar, "Annotated Control Flow Graph for Metamorphic Malware Detection," *The Computer Journal*, vol. 58, no. 10, pp. 2608–2621, Oct. 2015, ISSN: 0010-4620, 1460-2067. DOI: 10.1093/comjnl/bxu148. [Online]. Available: https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/bxu148 (visited on 09/04/2022).

[37] Y. Cao, Q. Miao, J. Liu, and L. Gao, "Abstracting minimal security-relevant behaviors for malware analysis," *Journal of Computer Virology and Hacking Techniques*, vol. 9, no. 4, pp. 193–204, Nov. 1, 2013, ISSN: 2263-8733. DOI: 10.1007/s11416-013-0186-3. [Online]. Available: https://doi.org/10.1007/s11416-013-0186-3 (visited on 05/24/2022).

[38] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, Jan. 1, 2011, ISSN: 0926-227X. DOI: 10.3233/JCS-2010-0410. [Online]. Available: http://content.iospress.com/articles/journal-of-computer-security/jcs410 (visited on 05/18/2022).

[39] G. Ramesh and A. Menen, "Automated dynamic approach for detecting ransomware using finite-state machine," *Decision Support Systems*, vol. 138, p. 113 400, Nov. 2020, ISSN: 01679236. DOI: 10.1016/j.dss.2020.113400. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016792362030155X (visited on 05/30/2022).

[40] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system," in *Proceedings of the 30th Annual Computer Security Applications Conference*, ser. ACSAC '14, New York, NY, USA: Association for Computing Machinery, Dec. 8, 2014, pp. 386–395, ISBN: 978-1-4503-3005-3. DOI: 10.1145/2664243.2664252. [Online]. Available: http://doi.org/10.1145/2664243.2664252 (visited on 05/17/2022).

[41] Sudhakar and S. Kumar, "An emerging threat Fileless malware: A survey and research challenges," *Cybersecurity*, vol. 3, no. 1, p. 1, Jan. 14, 2020, ISSN: 2523-3246. DOI: 10.1186/s42400-019-0043-x. [Online]. Available: https://doi.org/10.1186/s42400-019-0043-x (visited on 05/30/2022).

[42] B. Yu, Y. Fang, Q. Yang, Y. Tang, and L. Liu, "A survey of malware behavior description and analysis," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 5, pp. 583–603, May 1, 2018, ISSN: 2095-9230. DOI: 10.1631/FITEE.1601745. [Online]. Available: https://doi.org/10.1631/FITEE.1601745 (visited on 06/15/2022).

[43] L. Wang, B. Wang, J. Liu, Q. Miao, and a. J. Zhang, "Cuckoo-based Malware Dynamic Analysis," *International Journal of Performability Engineering*, vol. 15, no. 3, p. 772, Mar. 20, 2019, ISSN: 0973-1318. DOI: 10.23940/ijpe.19.03.p6.772781. [Online]. Available: http://www.ijpe-online.com/EN/10.23940/ijpe.19.03.p6.772781 (visited on 05/17/2022).

[44] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, and G. Vigna, "Neurlux: Dynamic malware analysis without feature engineering," in *Proceedings of the 35th Annual Computer Security Applications Conference*, ser. ACSAC '19, New York, NY, USA: Association for Computing Machinery, Dec. 9, 2019, pp. 444–455, ISBN: 978-1-4503-7628-0. DOI: 10.1145/3359789.3359835. [Online]. Available: https://doi.org/10.1145/3359789.3359835 (visited on 05/30/2022).

[45] Q. Wang, W. U. Hassan, D. Li, *et al.*, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis," in *Proceedings 2020 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2020, ISBN: 978-1-891562-61-7. DOI: 10.14722/ndss.2020.24167. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24167.pdf (visited on 05/30/2022).

[46] R. Jordaney, K. Sharad, S. K. Dash, *et al.*, "Transcend: Detecting Concept Drift in Malware Classification Models," presented at the 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 625–642, ISBN: 978-1-931971-40-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney (visited on 06/01/2022).

[47] "Machine Learning in Cybersecurity — Kaspersky." (), [Online]. Available: https://www.kaspersky.com/enterprise-security/wiki-section/products/machine-learning-in-cybersecurity (visited on 05/30/2022).

[48] X. Zhang, Y. Zhang, M. Zhong, *et al.*, "Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA: Association for Computing Machinery, Oct. 30, 2020, pp. 757–770, ISBN: 978-1-4503-7089-9. [Online]. Available: https://doi.org/10.1145/3372297.3417291 (visited on 05/30/2022).

[49] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "{TESSERACT}: Eliminating Experimental Bias in Malware Classification across Space and Time," presented at the 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 729–746, ISBN: 978-1-939133-06-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury (visited on 06/01/2022).

[50] T. Shields, "Anti-Debugging – A Developers View," p. 15, 2010.

[51] H. Shi, A. Alwabel, and J. Mirkovic, "Cardinal Pill Testing of System Virtual Machines," presented at the 23rd USENIX Security Symposium (USENIX Security 14), 2014, pp. 271–285, ISBN: 978-1-931971-15-7. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/shi (visited on 05/30/2022).

[52] O. Ferrand, "How to detect the Cuckoo Sandbox and to Strengthen it?" *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 51–58, Feb. 1, 2015, ISSN: 2263-8733. DOI: 10.1007/s11416-014-0224-9. [Online]. Available: https://doi.org/10.1007/s11416-014-0224-9 (visited on 05/17/2022).

[53] D. Rabadi and S. G. Teo, "Advanced Windows Methods on Malware Detection and Classification," in *Annual Computer Security Applications Conference*, ser. ACSAC '20, New York, NY, USA: Association for Computing Machinery, Dec. 7, 2020, pp. 54–68, ISBN: 978-1-4503-8858-0. DOI: 10.1145/3427228.3427242. [Online]. Available: https://doi.org/10.1145/3427228.3427242 (visited on 05/30/2022).

[54] J. Stiborek, T. Pevný, and M. Rehák, "Probabilistic analysis of dynamic malware traces," *Computers & Security*, vol. 74, pp. 221–239, May 2018, ISSN: 01674048. DOI: 10.1016/j.cose.2018.01.012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404818300336 (visited on 07/06/2022).

[55] A. Küchler, A. Mantovani, Y. Han, L. Bilge, and D. Balzarotti, "Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes," in *Proceedings 2021 Network and Distributed System Security Symposium*, Virtual: Internet Society, 2021, ISBN: 978-1-891562-66-2. DOI: 10.14722/ndss.2021.24475. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_4C-5_24475_paper.pdf (visited on 05/17/2022).

[56] E. Avllazagaj, Z. Zhu, L. Bilge, D. Balzarotti, and T. Dumitraș, "When Malware Changed Its Mind: An Empirical Study of Variable Program Behaviors in the Real World," presented at the 30th USENIX Security Symposium (USENIX Security 21), 2021, pp. 3487–3504, ISBN: 978-1-939133-24-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/avllazagaj (visited on 05/30/2022).

[57] "Hexacorn — Blog Beyond good ol' Run key – All parts." (), [Online]. Available: https://www.hexacorn.com/blog/2017/01/28/beyond-good-ol-run-key-all-parts/ (visited on 04/06/2023).

[58] "Persistence-info.github.io," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/ (visited on 04/06/2023).

[59] S. Garfinkel, A. J. Nelson, and J. Young, "A general strategy for differential forensic analysis," *Digital Investigation*, The Proceedings of the Twelfth Annual DFRWS Conference, vol. 9, S50–S59, Aug. 1, 2012, ISSN: 1742-2876. DOI: 10.1016/j.diin.2012.05.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S174228761200028X (visited on 04/11/2023).

[60] T. Teller and A. Hayon, “Enhancing Automated Malware Analysis Machines with Memory Analysis,”

[61] markruss. “Autoruns for Windows - Windows Sysinternals.” (), [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/autoruns (visited on 06/15/2022).

[62] last - @last0x00, *PersistenceSniper*, Apr. 3, 2023. [Online]. Available: https://github.com/last-byte/PersistenceSniper (visited on 04/06/2023).

[63] “Cuckoo Sandbox - Automated Malware Analysis.” (), [Online]. Available: https://cuckoosandbox.org/ (visited on 04/06/2023).

[64] *Sigma*, Sigma, Feb. 9, 2023. [Online]. Available: https://github.com/SigmaHQ/sigma (visited on 02/10/2023).

[65] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, “Malware Detection Based on Deep Learning of Behavior Graphs,” *Mathematical Problems in Engineering*, vol. 2019, e8195395, Feb. 11, 2019, ISSN: 1024-123X. DOI: 10.1155/2019/8195395. [Online]. Available: https://www.hindawi.com/journals/mpe/2019/8195395/ (visited on 05/29/2022).

[66] S. Alam, R. N. Horspool, I. Traore, and I. Sogukpinar, “A framework for metamorphic malware analysis and real-time detection,” *Computers & Security*, vol. 48, pp. 212–233, Feb. 1, 2015, ISSN: 0167-4048. DOI: 10.1016/j.cose.2014.10.011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404814001576 (visited on 07/14/2022).

[67] H. Lu, B. Zhao, J. Su, and P. Xie, “Generating Lightweight Behavioral Signature for Malware Detection in People-Centric Sensing,” *Wireless Personal Communications*, vol. 75, no. 3, pp. 1591–1609, Apr. 1, 2014, ISSN: 1572-834X. DOI: 10.1007/s11277-013-1400-9. [Online]. Available: https://doi.org/10.1007/s11277-013-1400-9 (visited on 01/30/2023).

[68] J. C. M. Baeten, “A brief history of process algebra,” *Theoretical Computer Science*, Process Algebra, vol. 335, no. 2, pp. 131–146, May 23, 2005, ISSN: 0304-3975. DOI: 10.1016/j.tcs.2004.07.036. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397505000307 (visited on 04/12/2023).

[69] I. Kirillov and P. Chase, “Malware Attribute Enumeration and Characterization,” p. 23, 2011.

[70] K. Oosthoek and C. Doerr, “SoK: ATT&CK Techniques and Trends in Windows Malware,” in *Security and Privacy in Communication Networks*, S. Chen, K.-K. R. Choo, X. Fu, W. Lou, and A. Mohaisen, Eds., ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Cham: Springer International Publishing, 2019, pp. 406–425, ISBN: 978-3-030-37228-6. DOI: 10.1007/978-3-030-37228-6_20.

[71] Admin. “Persistence: “the continued or prolonged existence of something”: Part 3 – WMI Event Subscription,” MDSec. (May 29, 2019), [Online]. Available: https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-3-wmi-event-subscription/ (visited on 05/21/2023).

[72] “FuzzySecurity — Windows Userland Persistence Fundamentals.” (), [Online]. Available: https://fuzzysecurity.com/tutorials/19.html (visited on 05/21/2023).

[73] “Scheduling Callbacks with WMI in C++ - wumb0in’.” (), [Online]. Available: https://wumb0.in/scheduling-callbacks-with-wmi-in-cpp.html (visited on 05/21/2023).

[74] “Event Triggered Execution: Windows Management Instrumentation Event Subscription, Sub-technique T1546.003 - Enterprise — MITRE ATT&CK®.” (), [Online]. Available: https://attack.mitre.org/techniques/T1546/003/ (visited on 05/21/2023).

[75] “Persistence and Privilege Escalation on Windows via Windows Management Instrumentation Event Subscription.” (Apr. 18, 2023), [Online]. Available: https://stmxcsr.com/persistence/wmi-persistence.html (visited on 05/21/2023).

[76] M. Graeber, “Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asyncronous, and Fileless Backdoor,”

[77] D. Perez. “Tales of a Threat Hunter 2.” (), [Online]. Available: https://eideon.com/2018-03-02-THL03-WMIBackdoors/ (visited on 05/21/2023).

[78] "Back in a Bit: Attacker Use of the Windows Background Intelligent Transfer Service," Mandiant. (), [Online]. Available: https://www.mandiant.com/resources/blog/attacker-use-of-windows-background-intelligent-transfer-service (visited on 05/21/2023).

[79] N. Noll. "BITS Persistence for Script Kiddies," TrustedSec. (Jun. 29, 2021), [Online]. Available: https://www.trustedsec.com/blog/bits-persistence-for-script-kiddies/ (visited on 05/21/2023).

[80] "BITS Jobs, Technique T1197 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1197/ (visited on 05/21/2023).

[81] K. Hayashi. "UBoatRAT Navigates East Asia," Unit 42. (Nov. 28, 2017), [Online]. Available: https://unit42.paloaltonetworks.com/unit42-uboatrat-navigates-east-asia/ (visited on 05/21/2023).

[82] "In-depth Malware Analysis: Malware Lingers with BITS." (Jun. 6, 2016), [Online]. Available: https://www.secureworks.com/blog/malware-lingers-with-bits (visited on 05/21/2023).

[83] "Cmd.exe AutoRun," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/cmdautorun.html (visited on 05/21/2023).

[84] cocomelonc. "Malware development: Persistence - part 5. AppInit_DLLs. Simple C++ example.," cocomelonc. (May 16, 2022), [Online]. Available: https://cocomelonc.github.io/tutorial/2022/05/16/malware-pers-5.html (visited on 05/21/2023).

[85] stevewhims. "AppInit DLLs and Secure Boot - Win32 apps." (Sep. 29, 2022), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/dlls/secure-boot-and-appinit-dlls (visited on 05/21/2023).

[86] "Event Triggered Execution: AppInit DLLs, Sub-technique T1546.010 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/010/ (visited on 05/21/2023).

[87] "Event Triggered Execution: Screensaver, Sub-technique T1546.002 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/002/ (visited on 05/21/2023).

[88] "Grandoreiro Malware Now Targeting Banks in Spain." (), [Online]. Available: https://securityintelligence.com/posts/grandoreiro-malware-now-targeting-banks-in-spain/ (visited on 05/21/2023).

[89] "Modifying .lnk Shortcuts." (), [Online]. Available: https://www.ired.team/offensive-security/persistence/modifying-.lnk-shortcuts (visited on 05/21/2023).

[90] "Browser Extensions, Technique T1176 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1176/ (visited on 05/21/2023).

[91] M. Frisbie. "Let's build a Chrome extension that steals everything," Building Browser Extensions. (Feb. 21, 2023), [Online]. Available: https://mattfrisbie.substack.com/p/spy-chrome-extension (visited on 05/21/2023).

[92] "Windows Terminal Profile," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/windowsterminalprofile.html (visited on 05/21/2023).

[93] K. Almansa. "DLL Proxying," InfoSec Blog. (Jul. 13, 2017), [Online]. Available: https://kevinalmansa.github.io/application%20security/DLL-Proxying/ (visited on 04/12/2023).

[94] Deland-Han. "Windows registry for advanced users - Windows Server." (Mar. 9, 2023), [Online]. Available: https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users (visited on 03/29/2023).

[95] A. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th edition. Boston: Pearson, Mar. 10, 2014, 1136 pp., ISBN: 978-0-13-359162-0.

[96] O. Ben-Kiki, "YAML Ain't Markup Language (YAML™) Version 1.1,"

[97] H. Klein. "Active Setup Explained • Helge Klein," Helge Klein. (Apr. 22, 2010), [Online]. Available: https://helgeklein.com/blog/active-setup-explained/ (visited on 05/21/2023).

[98] "Boot or Logon Autostart Execution: Active Setup, Sub-technique T1547.014 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1547/014/ (visited on 05/21/2023).

[99] "Pin - A Dynamic Binary Instrumentation Tool," Intel. (), [Online]. Available: `https://www.intel.com/content/www/us/en/developer/articles/tool/pin-a-dynamic-binary-instrumentation-tool.html` (visited on 03/31/2023).

[100] L. Maffia, D. Nisi, P. Kotzias, G. Lagorio, S. Aonzo, and D. Balzarotti. "Longitudinal Study of the Prevalence of Malware Evasive Techniques." arXiv: `2112.11289 [cs]`. (Dec. 21, 2021), [Online]. Available: `http://arxiv.org/abs/2112.11289` (visited on 03/31/2023), preprint.

[101] "VirusTotal - Home." (), [Online]. Available: `https://www.virustotal.com/gui/home/upload` (visited on 05/16/2023).

[102] M. Lab, *AVClass*, May 12, 2023. [Online]. Available: `https://github.com/malicialab/avclass` (visited on 05/19/2023).

[103] "Hiding Your Syscalls," PassTheHashBrowns. (Jun. 9, 2021), [Online]. Available: `https://passthehashbrowns.github.io/hiding-your-syscalls` (visited on 04/04/2023).

[104] O. Moe. "Persistence using Universal Windows Platform apps (APPX)," Oddvar Moe's Blog. (Sep. 6, 2018), [Online]. Available: `https://oddvar.moe/2018/09/06/persistence-using-universal-windows-platform-apps-appx/` (visited on 05/21/2023).

[105] "AeDebug," persistence-info.github.io. (), [Online]. Available: `https://persistence-info.github.io/Data/aedebug.html` (visited on 05/21/2023).

[106] Karl-Bridge-Microsoft. "Configuring Automatic Debugging - Win32 apps." (Jan. 6, 2021), [Online]. Available: `https://learn.microsoft.com/en-us/windows/win32/debug/configuring-automatic-debugging` (visited on 05/21/2023).

[107] "Event Triggered Execution: Change Default File Association, Sub-technique T1546.001 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: `https://attack.mitre.org/techniques/T1546/001/` (visited on 05/21/2023).

[108] "Changing Default File Associations in Windows 10 and 11," Windows OS Hub. (Sep. 29, 2022), [Online]. Available: `https://woshub.com/managing-default-file-associations-in-windows-10/` (visited on 05/21/2023).

[109] "SolarMarker campaign used novel registry changes to establish persistence," Sophos News. (Feb. 1, 2022), [Online]. Available: `https://news.sophos.com/en-us/2022/02/01/solarmarker-campaign-used-novel-registry-changes-to-establish-persistence/` (visited on 05/21/2023).

[110] "Explorer tools," persistence-info.github.io. (), [Online]. Available: `https://persistence-info.github.io/Data/explorertools.html` (visited on 05/21/2023).

[111] "Hexacorn — Blog Beyond good ol' Run key, Part 55." (), [Online]. Available: `https://www.hexacorn.com/blog/2017/01/18/beyond-good-ol-run-key-part-55/` (visited on 05/21/2023).

[112] *Sigma*, Sigma, May 21, 2023. [Online]. Available: `https://github.com/SigmaHQ/sigma/blob/1a57509e858674a5acf297061d64b3ffe59e6b3d/rules/windows/registry/registry_set/registry_set_persistence_mycomputer.yml` (visited on 05/21/2023).

[113] "Hexacorn — Blog Beyond good ol' Run key, Part 4." (), [Online]. Available: `https://www.hexacorn.com/blog/2013/09/19/beyond-good-ol-run-key-part-4/` (visited on 05/21/2023).

[114] Mikejo5000. "Debug using the Just-In-Time Debugger - Visual Studio (Windows)." (Jan. 14, 2023), [Online]. Available: `https://learn.microsoft.com/en-us/visualstudio/debugger/debug-using-the-just-in-time-debugger` (visited on 05/21/2023).

[115] O. Moe. "Persistence using RunOnceEx – Hidden from Autoruns.exe," Oddvar Moe's Blog. (Mar. 21, 2018), [Online]. Available: `https://oddvar.moe/2018/03/21/persistence-using-runonceex-hidden-from-autoruns-exe/` (visited on 05/21/2023).

[116] O. Moe. "Persistence using GlobalFlags in Image File Execution Options – Hidden from Autoruns.exe," Oddvar Moe's Blog. (Apr. 10, 2018), [Online]. Available: `https://oddvar.moe/2018/04/10/persistence-using-`

globalflags-in-image-file-execution-options-hidden-from-autoruns-exe/ (visited on 05/21/2023).

[117] "Monitoring Silent Process Exit," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/silentexitmonitor.html (visited on 05/21/2023).

[118] stevewhims. "Task Scheduler for developers - Win32 apps." (Feb. 8, 2023), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page (visited on 05/21/2023).

[119] "Task Scheduler," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/taskscheduler.html (visited on 05/21/2023).

[120] "FuzzySecurity — Windows Userland Persistence Fundamentals." (), [Online]. Available: https://fuzzysecurity.com/tutorials/19.html (visited on 05/21/2023).

[121] "Persistence 101: Looking at the Scheduled Tasks." (Apr. 18, 2023), [Online]. Available: https://stmxcsr.com/persistence/scheduled-tasks.html (visited on 05/21/2023).

[122] giuliocomi, *Backoori*, Apr. 11, 2023. [Online]. Available: https://github.com/giuliocomi/backoori (visited on 05/21/2023).

[123] "Abusing Windows 10 Narrator's 'Feedback-Hub' URI for Fileless Persistence," Abusing Windows 10 Narrator's 'Feedback-Hub' URI for Fileless Persistence. (), [Online]. Available: https://giuliocomi.blogspot.com/2019/10/abusing-windows-10-narrators-feedback.html (visited on 05/21/2023).

[124] *Sigma*, Sigma, May 21, 2023. [Online]. Available: https://github.com/SigmaHQ/sigma/blob/b4cb047ae720b37b11f8506de7965dc29d5920be/rules/windows/registry/registry_event/registry_event_narrator_feedback_persistance.yml (visited on 05/21/2023).

[125] "Boot or Logon Initialization Scripts: Logon Script (Windows), Sub-technique T1037.001 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1037/001/ (visited on 05/21/2023).

[126] N. Noll. "Abusing Windows Telemetry for Persistence," TrustedSec. (Jun. 9, 2020), [Online]. Available: https://www.trustedsec.com/blog/abusing-windows-telemetry-for-persistence/ (visited on 05/21/2023).

[127] "TelemetryController," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/telemetrycontroller.html (visited on 05/21/2023).

[128] "AMSI Providers," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/amsi.html (visited on 05/21/2023).

[129] alvinashcraft. "Developer audience, and sample code - Win32 apps." (Jan. 29, 2020), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/amsi/dev-audience (visited on 05/21/2023).

[130] "PSBits/FakeAMSI.c at master · gtworek/PSBits," GitHub. (), [Online]. Available: https://github.com/gtworek/PSBits (visited on 05/21/2023).

[131] "Hexacorn — Blog Beyond good ol' Run key, Part 3." (), [Online]. Available: https://www.hexacorn.com/blog/2013/01/19/beyond-good-ol-run-key-part-3/ (visited on 05/21/2023).

[132] "Persistence Techniques That Persist." (), [Online]. Available: https://www.cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist (visited on 05/21/2023).

[133] "Event Triggered Execution: AppCert DLLs, Sub-technique T1546.009 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/009/ (visited on 05/21/2023).

[134] *Atomic Red Team*, Red Canary, May 21, 2023. [Online]. Available: https://github.com/redcanaryco/atomic-red-team/blob/ebdec5d757a617c349021bc86132634f8d013a0e/atomics/T1546.009/T1546.009.md (visited on 05/21/2023).

[135] "Prevent Bypass of AppLocker and SAFER alias Software Restriction Policies." (), [Online]. Available: https://skanthak.homepage.t-online.de/appcert.html (visited on 05/21/2023).

[136] "Event Triggered Execution: Application Shimming, Sub-technique T1546.011 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/011/ (visited on 05/21/2023).

[137] "Application Shimming." (), [Online]. Available: https://www.ired.team/offensive-security/persistence/t1138-application-shimming (visited on 05/21/2023).

[138] *Sigma*, Sigma, May 21, 2023. [Online]. Available: https://github.com/SigmaHQ/sigma/blob/33b370d49bd6aed85bd23827aa16a50bd06d691a/rules/windows/registry/registry_set/registry_set_shim_databases_persistence.yml (visited on 05/21/2023).

[139] "To SDB, Or Not To SDB: FIN7 Leveraging Shim Databases for Persistence," Mandiant. (), [Online]. Available: https://www.mandiant.com/resources/blog/fin7-shim-databases-persistence (visited on 05/21/2023).

[140] "Autodial DLL," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/autodialdll.html (visited on 05/21/2023).

[141] "Hexacorn — Blog Beyond good ol' Run key, Part 24." (), [Online]. Available: https://www.hexacorn.com/blog/2015/01/13/beyond-good-ol-run-key-part-24/ (visited on 05/21/2023).

[142] stevewhims. "AutoDial Connection Operations - Win32 apps." (Aug. 19, 2020), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/rras/autodial-connection-operations (visited on 05/21/2023).

[143] "Hexacorn — Blog Beyond good ol' Run key, Part 114." (), [Online]. Available: https://www.hexacorn.com/blog/2019/09/07/beyond-good-ol-run-key-part-114/ (visited on 05/21/2023).

[144] jwmsft. "How to Register a Handler for a Device Event - Win32 apps." (Jan. 7, 2021), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/shell/how-to-register-a-handler-for-a-device-event (visited on 05/21/2023).

[145] "Persistence Techniques That Persist." (), [Online]. Available: https://www.cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist (visited on 05/21/2023).

[146] cocomelonc. "Malware development: Persistence - part 3. COM DLL hijack. Simple C++ example.," cocomelonc. (May 2, 2022), [Online]. Available: https://cocomelonc.github.io/tutorial/2022/05/02/malware-pers-3.html (visited on 05/21/2023).

[147] Admin. "Persistence: "the continued or prolonged existence of something": Part 2 - COM Hijacking," MDSec. (May 26, 2019), [Online]. Available: https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-2-com-hijacking/ (visited on 05/21/2023).

[148] "Event Triggered Execution: Component Object Model Hijacking, Sub-technique T1546.015 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/015/ (visited on 05/21/2023).

[149] "Persistence: Component Object Model (COM) hijacking." (Apr. 18, 2023), [Online]. Available: https://stmxcsr.com/persistence/com-hijacking.html (visited on 05/21/2023).

[150] *Sigma*, Sigma, May 21, 2023. [Online]. Available: https://github.com/SigmaHQ/sigma/blob/16990093933fe8c82c3ad879d306d706b350162a/rules/windows/registry/registry_set/registry_set_persistence_search_order.yml (visited on 05/21/2023).

[151] susannah.matt@redcanary.com. "Detecting COR_PROFILER manipulation for persistence," Red Canary. (), [Online]. Available: https://redcanary.com/blog/cor_profiler-for-persistence/ (visited on 05/21/2023).

[152] "Hijack Execution Flow: COR_PROFILER, Sub-technique T1574.012 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1574/012/ (visited on 05/21/2023).

[153] "Code Signing DLL," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/codesigning.html (visited on 05/21/2023).

[154] *Sigma*, Sigma, May 21, 2023. [Online]. Available: https://github.com/SigmaHQ/sigma/blob/16990093933fe8c82c3ad879d306d706b350162a/rules/windows/registry/registry_set/registry_set_sip_persistence.yml (visited on 05/21/2023).

[155] M. Graeber, "Subverting Trust in Windows,"

[156] "PSBits/SIP at master · gtworek/PSBits," GitHub. (), [Online]. Available: https://github.com/gtworek/PSBits (visited on 05/21/2023).

[157] Administrator. "Persistence – Context Menu," Penetration Testing Lab. (Mar. 13, 2023), [Online]. Available: https://pentestlab.blog/2023/03/13/persistence-context-menu/ (visited on 05/21/2023).

[158] "Hijack Explorer Context Menu for Persistence & Fun," ristbs's blog. (Feb. 15, 2023), [Online]. Available: https://ristbs.github.io/2023/02/15/hijack-explorer-context-menu-for-persistence-and-fun.html (visited on 05/21/2023).

[159] RistBS, *ContextMenuHijack*, May 19, 2023. [Online]. Available: https://github.com/RistBS/ContextMenuHijack (visited on 05/21/2023).

[160] "Credential Manager DLL," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/credmandll.html (visited on 05/21/2023).

[161] "PSBits/PasswordStealing/NPPSpy at master · gtworek/PSBits," GitHub. (), [Online]. Available: https://github.com/gtworek/PSBits (visited on 05/21/2023).

[162] "Disk Cleanup Handler," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/diskcleanuphandler.html (visited on 05/21/2023).

[163] "Hexacorn — Blog Beyond good ol' Run key, Part 86." (), [Online]. Available: https://www.hexacorn.com/blog/2018/09/02/beyond-good-ol-run-key-part-86/ (visited on 05/21/2023).

[164] QuinnRadich. "Creating a Disk Cleanup Handler - Win32 apps." (Aug. 20, 2021), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/lwef/disk-cleanup (visited on 05/21/2023).

[165] "Image File Execution Options," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/ifeo.html (visited on 05/21/2023).

[166] P. Arntz. "An Introduction to Image File Execution Options — Malwarebytes Labs," Malwarebytes. (Dec. 4, 2015), [Online]. Available: https://www.malwarebytes.com/blog/news/2015/12/an-introduction-to-image-file-execution-options (visited on 05/21/2023).

[167] O. Moe. "Persistence using GlobalFlags in Image File Execution Options – Hidden from Autoruns.exe," Oddvar Moe's Blog. (Apr. 10, 2018), [Online]. Available: https://oddvar.moe/2018/04/10/persistence-using-globalflags-in-image-file-execution-options-hidden-from-autoruns-exe/ (visited on 05/21/2023).

[168] "Event Triggered Execution: Image File Execution Options Injection, Sub-technique T1546.012 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/012/ (visited on 05/21/2023).

[169] "Authentication Packages," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/authenticationpackages.html (visited on 05/21/2023).

[170] "Boot or Logon Autostart Execution: Authentication Package, Sub-technique T1547.002 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1547/002/ (visited on 05/21/2023).

[171] "LSA Extension," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/lsaaextension.html (visited on 05/21/2023).

[172] "Password Filter," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/passwordfilter.html (visited on 05/21/2023).

[173] "Boot or Logon Autostart Execution: Security Support Provider, Sub-technique T1547.005 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1547/005/ (visited on 05/21/2023).

[174] "MPNotify," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/mpnotify.html (visited on 05/21/2023).

[175] "Boot or Logon Autostart Execution: Winlogon Helper DLL, Sub-technique T1547.004 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1547/004/ (visited on 05/21/2023).

[176] "Windows Registry Persistence, Part 2: The Run Keys and Search-Order." (Oct. 6, 2020), [Online]. Available: https://web.archive.org/web/20201006201906/https://blogs.blackberry.com/en/2013/09/windows-registry-persistence-part-2-the-run-keys-and-search-order (visited on 05/21/2023).

[177] ".NET Startup Hooks," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/dotnetstartuphooks.html (visited on 05/21/2023).

[178] *.NET Runtime*, .NET Platform, May 21, 2023. [Online]. Available: https://github.com/dotnet/runtime/blob/72fb58b3dfd4f9a40d5f3b0f87e26d9f24136570/docs/design/features/host-startup-hook.md (visited on 05/21/2023).

[179] "Natural Language 6 DLLs," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/naturallanguage6.html (visited on 05/21/2023).

[180] "Hexacorn — Blog Beyond good ol' Run key, Part 98." (), [Online]. Available: https://www.hexacorn.com/blog/2018/12/30/beyond-good-ol-run-key-part-98/ (visited on 05/21/2023).

[181] Administrator. "Persistence – Netsh Helper DLL," Penetration Testing Lab. (Oct. 29, 2019), [Online]. Available: https://pentestlab.blog/2019/10/29/persistence-netsh-helper-dll/ (visited on 05/21/2023).

[182] "Event Triggered Execution: Netsh Helper DLL, Sub-technique T1546.007 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/007/ (visited on 05/21/2023).

[183] "Hijack Execution Flow: Path Interception by PATH Environment Variable, Sub-technique T1574.007 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1574/007/ (visited on 05/21/2023).

[184] "Persistence and Privilege Escalation on Windows via Print Monitors." (Apr. 18, 2023), [Online]. Available: https://stmxcsr.com/persistence/print-monitor.html (visited on 05/21/2023).

[185] cocomelonc. "Malware development: Persistence - part 8. Port monitors. Simple C++ example.," cocomelonc. (Jun. 19, 2022), [Online]. Available: https://cocomelonc.github.io/tutorial/2022/06/19/malware-pers-8.html (visited on 05/21/2023).

[186] hickeys. "AddMonitor function (Winspool.h) - Win32 apps." (Jan. 7, 2021), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/printdocs/addmonitor (visited on 05/21/2023).

[187] "Boot or Logon Autostart Execution: Port Monitors, Sub-technique T1547.010 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1547/010/ (visited on 05/21/2023).

[188] "Persistence and Privilege Escalation on Windows via Print Processors." (Apr. 18, 2023), [Online]. Available: https://stmxcsr.com/persistence/print-processor.html (visited on 05/21/2023).

[189] "Boot or Logon Autostart Execution: Print Processors, Sub-technique T1547.012 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1547/012/ (visited on 05/21/2023).

[190] "Persistence Techniques That Persist." (), [Online]. Available: https://www.cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist (visited on 05/21/2023).

[191] "Attacking RDP from Inside: How we abused named pipes for smart-card hijacking, unauthorized file system access to client machines and more." (), [Online]. Available: https://www.cyberark.com/resources/threat-research-blog/attacking-rdp-from-inside (visited on 05/21/2023).

[192] O. Moe. “Persistence using RunOnceEx – Hidden from Autoruns.exe,” Oddvar Moe’s Blog. (Mar. 21, 2018), [Online]. Available: https://oddvar.moe/2018/03/21/persistence-using-runonceex-hidden-from-autoruns-exe/ (visited on 05/21/2023).

[193] “ServerLevelPluginDll,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/serverlevelplugindll.html (visited on 05/21/2023).

[194] “TS Initial Program,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/tsinitialprogram.html (visited on 05/21/2023).

[195] “Boot or Logon Autostart Execution: Time Providers, Sub-technique T1547.003 - Enterprise — MITRE ATT&CK®.” (), [Online]. Available: https://attack.mitre.org/techniques/T1547/003/ (visited on 05/21/2023).

[196] stevewhims. “Time Provider - Win32 apps.” (Jan. 7, 2021), [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/sysinfo/time-provider (visited on 05/21/2023).

[197] “Persistence and Privilege Escalation on Windows via Time Providers.” (Apr. 18, 2023), [Online]. Available: https://stmxcsr.com/persistence/time-provider.html (visited on 05/21/2023).

[198] “Hexacorn — Blog Beyond good ol’ Run key, Part 116.” (), [Online]. Available: https://www.hexacorn.com/blog/2019/09/20/beyond-good-ol-run-key-part-116/ (visited on 05/21/2023).

[199] “HKCU Load,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/windowsload.html (visited on 05/21/2023).

[200] “Group Policy Client Side Extension,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/gpoextension.html (visited on 05/21/2023).

[201] R. markl. “Creating a Policy Callback Function.” (May 31, 2018), [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/windows/desktop/policy/creating-a-policy-callback-function (visited on 05/21/2023).

[202] “Hexacorn — Blog Beyond good ol’ Run key, Part 92.” (), [Online]. Available: https://www.hexacorn.com/blog/2018/10/11/beyond-good-ol-run-key-part-92/ (visited on 05/21/2023).

[203] “.chm helper DLL,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/htmlhelpauthor.html (visited on 05/21/2023).

[204] “Hexacorn — Blog Beyond good ol’ Run key, Part 76.” (), [Online]. Available: https://www.hexacorn.com/blog/2018/04/22/beyond-good-ol-run-key-part-76/ (visited on 05/21/2023).

[205] “Hexacorn — Blog Beyond good ol’ Run key, Part 77.” (), [Online]. Available: https://www.hexacorn.com/blog/2018/04/23/beyond-good-ol-run-key-part-77/ (visited on 05/21/2023).

[206] Administrator. “Persistence – Accessibility Features,” Penetration Testing Lab. (Nov. 13, 2019), [Online]. Available: https://pentestlab.blog/2019/11/13/persistence-accessibility-features/ (visited on 05/21/2023).

[207] “Event Triggered Execution: Accessibility Features, Sub-technique T1546.008 - Enterprise — MITRE ATT&CK®.” (), [Online]. Available: https://attack.mitre.org/techniques/T1546/008/ (visited on 05/21/2023).

[208] “Hexacorn — Blog Beyond good ol’ Run key, Part 135.” (), [Online]. Available: https://www.hexacorn.com/blog/2022/01/16/beyond-good-ol-run-key-part-135/ (visited on 05/21/2023).

[209] T. I. Team. “Colibri Loader combines Task Scheduler and PowerShell in clever persistence technique,” Malwarebytes. (Apr. 5, 2022), [Online]. Available: https://www.malwarebytes.com/blog/threat-intelligence/2022/04/colibri-loader-combines-task-scheduler-and-powershell-in-clever-persistence-technique (visited on 05/21/2023).

[210] “PowerShell Profile,” persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/powershellprofile.html (visited on 05/21/2023).

[211] "Event Triggered Execution: PowerShell Profile, Sub-technique T1546.013 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1546/013/ (visited on 05/21/2023).

[212] "Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder, Subtechnique T1547.001 - Enterprise — MITRE ATT&CK®." (), [Online]. Available: https://attack.mitre.org/techniques/T1547/001/ (visited on 05/21/2023).

[213] "Startup Folder," persistence-info.github.io. (), [Online]. Available: https://persistence-info.github.io/Data/startupfolder.html (visited on 05/21/2023).

[214] James. "Persistence with KeePass - Part 1," Medium. (Jun. 30, 2019), [Online]. Available: https://medium.com/@two06/persistence-with-keepass-part-1-d2e705326aa6 (visited on 05/21/2023).

[215] "Persistence with KeePass -Part 2. In part 1 we saw how we can use KeePass... — by James — Medium." (), [Online]. Available: https://medium.com/@two06/persistence-with-keepass-part-2-3e328b24e117 (visited on 05/21/2023).

[216] cybleinc. "Targeted Attacks being carried out via DLL SideLoading," Cyble. (Jul. 27, 2022), [Online]. Available: https://blog.cyble.com/2022/07/27/targeted-attacks-being-carried-out-via-dll-sideloading/ (visited on 05/21/2023).

[217] *Sigma*, Sigma, May 21, 2023. [Online]. Available: https://github.com/SigmaHQ/sigma/blob/bc2188c72bc89469bf425ee3b1a174d58f5586ed/rules/windows/file_event/file_event_win_iphlpapi_dll_sideloading.yml (visited on 05/21/2023).

A Persistence techniques

	Requirements					Resulting persistence			
Name	**File System Write**	**Registry write**	**Elevated privileges**	**Interprocess communication**	**Additional Software**	**Elevated privileges**	**Moment of execution**	**Destructive**	**Execution type**
Databases									
Other									
BITS jobs [78]–[82]	✓	✗	✗	✓	✗	✗	P	✗	EwA
WMI subscriptions [71]–[77]	✓	✗	✓	O	✗	✓	E/P	✗	C
Registry									
APPX [104]	✗	✓	✗	✗	✗	✗	U	✓	E
Active Setup [97], [98]	✗	✓	✓	✗	✗	✗	B	✓	C
AeDebug registry key [105], [106]	✗	✓	✓	✗	✗	✗	E	✗	C
Change default file extensions [107]–[109]	✗	✓	✓	✗	✗	✗	U	✗	EwA
Explorer tools [110]–[112]	✗	✓	✓	✗	✗	✗	U	✗	C
Group policy logon script	✗	✓	✓	✗	✗	✓	B	✗	C
NET DbgManagedDebugger [113], [114]	✗	✓	✓	✗	✗	✗	E	✗	EwA
Run registry keys [20]	✗	✓	✗	✗	✗	✗	B	✗	C
RunonceEx exe [115]	✗	✓	✓	✗	✗	✗	B	✗	C
SilentProcessExit [116], [117]	✗	✓	✓	✗	✗	✗	U	✗	C
Task scheduler [118]–[121]	✗	✓	S	✗	✗	✓	P	✗	C
Universal App URI [122]–[124]	✗	✓	✗	✗	✗	✗	U	✗	EwA
User Init Mpr Logon Script [125]	✗	✓	✗	✗	✗	✗	B	✗	C
Windows telemetry [126], [127]	✗	✓	✓	✗	✗	✓	P	✗	C
cmd AutoRun [83]	✗	✓	✗	✗	✗	✗	U	✗	C
Databases and Filesystem									
Registry and Operating System Files									
AMSI provider [128]–[130]	✓	✓	✓	✗	✗	✗	U	✗	D
App paths [131], [132]	✓	✓	✗	✗	✗	✗	U	✗	E
AppInit DLL [84]–[86]	✓	✓	✓	✗	✗	✗	U	✗	D
Appcert DLL [133]–[135]	✓	✓	✓	✗	✗	✗	U	✗	D
Application shimming [136]–[139]	✓	✓	✓	✗	✗	✗	U	✗	D/E

	Requirements					**Resulting persistence**			
Name	**File System Write**	**Registry write**	**Elevated privileges**	**Interprocess communication**	**Additional Software**	**Elevated privileges**	**Moment of execution**	**Destructive**	**Execution type**
Autodial DLL [140]–[142]	✓	✓	✓	✗	✗	✗	U	✗	D
Autoplay handlers [143], [144]	✓	✓	✓	✗	✗	✗	U	✗	E
Boot verification program [145]	✓	✓	✓	✗	✗	✓	B	✗	E
COM hijack [146]–[150]	✓	✓	✗	✗	✗	✗	U	✗	D/E
COR_PROFILER [151], [152]	✓	✓	✗	✗	✗	✗	U	✗	D
Code signing DLL [153]–[156]	✓	✓	✓	✗	✗	✗	E	✗	D
Context Menu handler [157]–[159]	✓	✓	✗	✗	✗	✗	U	✗	D
Credential Manager DLL [160], [161]	✓	✓	✓	✗	✗	✓	B	✗	D
Disk Cleanup Handler [162]–[164]	✓	✓	✓	✗	✗	✗	U	✗	D
Filter handlers [27]	✓	✓	✓	✗	✗	✓	U	✗	D
Image file execution options [165]–[168]	✓	✓	✓	✗	✗	✗	U	✗	E
LSA authentication packages [169], [170]	✓	✓	✓	✗	✗	✗	B	✗	D
LSA extensions [171]	✓	✓	TI	✗	✗	✗	B	✗	D
LSA notification packages [172]	✓	✓	✓	✗	✗	✓	B	✗	D
LSA security packages [173]	✓	✓	✓	✗	✗	✗	B	✗	D
MPnotify winlogon [174]–[176]	✓	✓	✓	✗	✗	✓	B	✓	D/E
NET startup hooks [177], [178]	✓	✓	✗	✗	✓	✗	U	✗	D
Natual Language DLLs [179], [180]	✓	✓	✓	✗	✗	✓	B	✗	D
Netsh Helper DLL [181], [182]	✓	✓	✓	✗	✗	✗	E	✗	D
Path interception [183]	✓	✓	✓	✗	✗	✗	U	✗	E
Print monitors [184]–[187]	✓	✓	✓	✗	✗	✓	B	✗	D
Print processors [188], [189]	✓	✓	✓	✗	✗	✓	B	✗	D
RDP startup program [190], [191]	✓	✓	✓	✗	✗	✗	U	✗	E
RunonceEx DLL [192]	✓	✓	✓	✗	✗	✗	B	✗	D
RunonceEx depend DLL [192]	✓	✓	✓	✗	✗	✗	B	✗	D
Screensaver [87]	✓	✓	✗	✗	✗	✗	E	✗	E
Server level plugin DLL [193]	✓	✓	✓	✗	✓	✓	B	✗	D
Startup folder add [21], [22]	✓	✓	✗	✗	✗	✗	B	✗	E
TS Initial Program [194]	✓	✓	✗	✗	✗	✗	U	✗	E

	Requirements					**Resulting persistence**			
Name	**File System Write**	**Registry write**	**Elevated privileges**	**Interprocess communication**	**Additional Software**	**Elevated privileges**	**Moment of execution**	**Destructive**	**Execution type**
Time providers [195]–[197]	✓	✓	✓	✗	✗	✗	B	✗	D
Windows Error Reporting Debugger [198]	✓	✓	✓	✗	✗	✗	E	✗	E
Windows Load [199]	✓	✓	✗	✗	✗	✗	B	✗	E
Windows services [23]–[25]	✓	✓	✓	✗	✗	✓	B/P	✗	D/E
Winlogon GP Client-side extension [200]–[202]	✓	✓	✓	✗	✗	✓	E	✗	D
chm helper dll [203], [204]	✓	✓	✗	✗	✗	✗	U	✗	D
hhctrl [205]	✓	✓	TI	✗	✗	✗	U	✗	D
File system									
Operating System Files									
Accessibility Tools Backdoor [206], [207]	✓	✗	TI	✗	✗	✓	U	✗	E
DLL hijack [28]	✓	✗	✗	✗	✗	✗	U	✗	D
ErrorHandler cmd Hijack [208]	✓	✗	✓	✗	✗	✗	E	✗	C
Get-Variable [209]	✓	✗	✗	✗	✗	✗	U	✗	E
PowerShell profile [210], [211]	✓	✗	✗	✗	✗	✗	U	✗	C
Startup folder [212], [213]	✓	✗	✗	✗	✗	✗	B	✗	E
lnk shortcuts [88], [89]	✓	✗	✗	✗	✗	✗	U	✗	EwA
Additional software files									
Browser extensions [90], [91]	✓	✗	✗	✗	✓	✗	U	✗	S
Keepass plugins [214]	✓	✗	✓	✗	✓	✗	U	✗	D
Keepass trigger [215]	✓	✗	✗	✗	✓	✗	E	✗	C
Windows Terminal Profile [92]	✓	✗	✗	✗	✓	✗	U	✓	C
iphlpapi DLL [216], [217]	✓	✗	✗	✗	✓	✗	U	✗	D

Table 2: The persistence techniques that we identified and that meet our criteria. For the Elevated Privileges as a requirement, S = System privileges and TI = TrustedInstaller privileges. For Interprocess communication, O = optional. For the moment of executions, we have B=boot, U=user action, E=system event, P=periodically. For Code type we have C=command, E=exe, EwA=Exe with arguments, D=DLL, S=script for third party program .

B Persistence techniques

B.1 Accessibility Tools Backdoor

B.1.1 Origin of the technique

Windows contains multiple accessibility features that enable the use of the operating system by impaired users. These features can be accessed by some hotkeys. By hijacking the executables of the accessibility tools, the malware can execute code when the key combination is pressed.

B.1.2 How to achieve persistence

Replace accessibility features with a malicious file.

The accessibility tools are the following:

- On-Screen Keyboard: C:\Windows\System32\osk.exe
- Magnifier: C:\Windows\System32\Magnify.exe
- Narrator: C:\Windows\System32\Narrator.exe
- Display Switcher: C:\Windows\System32\DisplaySwitch.exe
- App Switcher: C:\Windows\System32\AtBroker.exe
- Sticky keys: C:\Windows\System32\sethc.exe
- Utilman: C:\Windows\System32\utilman.exe

B.1.3 Requirements

In recent windows versions execution is protected, as the executables have to be signed. Required Trusted installer privileges to replace the executables.

B.1.4 The achieved persistence

When the user tries to use one of the accessibility tools, the code is executed. If the tools is opened before the user logs in, the launched process has system privileges.

B.1.5 More information

- `https://attack.mitre.org/techniques/T1546/008/`
- `https://pentestlab.blog/2019/11/13/persistence-accessibility-features/`

B.2 Active Setup

B.2.1 Origin of the technique

Active Setup is a mechanism used to set up configuration when a new user logs in for the first time after new software has been installed. To know when to perform this action Active Setup checks if each machine part component GUID is present in the user part of the registry. If one is missing, Active Setup runs the commands associated with the missing GUID.

B.2.2 How to achieve persistence

By adding a new GUID to the Registry under HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components , it is possible to run a command at login. The command should be stored in StubPath.

B.2.3 Requirements

Writing to the HKLM registry requires administrator privileges. Also, not all systems seem to execute Active Setup automatically on boot. To still execute, it can be combined with another persistence technique by running %systemroot%\system32\runonce.exe /AlternateShellStartup.

B.2.4 The achieved persistence

This persistence technique executes the specified command when the user logs in, before the desktop appears. The desktop does not show before the command is finished. To achieve persistence after another reboot, removing the key in the user registry HKCU\SOFTWARE\Microsoft\Active Setup\Installed Components suffices.

B.2.5 More information

- `https://attack.mitre.org/techniques/T1547/014/`
- `https://helgeklein.com/blog/active-setup-explained/`

B.3 AeDebug registry key

B.3.1 Origin of the technique

Windows has a feature to automatically start a debugger when the system or an application crashes. This techniques runs the malicious code instead of the debugger.

B.3.2 How to achieve persistence

At the registry key values Auto=1 and Debugger = <full executable path> <additional parameters> to the key HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug or HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\AeDebug for 32-bit applications.

B.3.3 Requirements

Administrator privileges to write to the registry.

B.3.4 The achieved persistence

When an application or the system crashes, the executable is started.

B.3.5 More information

- `https://persistence-info.github.io/Data/aedebug.html`
- `https://docs.microsoft.com/en-us/windows/win32/debug/configuring-automatic-debugging`

B.4 AMSI provider

B.4.1 Origin of the technique

This technique is based on the Windows Antimalware Scan Interface (AMSI). AMSI is used to evaluate whether a script is malicious. Any program running can request AMSI to scan a script, for example PowerShell does this before running anything. Anti-malware solutions can register as a provider. For this technique the malware registers itself as an AMSI provider.

B.4.2 How to achieve persistence

First, you need to register the COM server by adding a new CLSID in HKLM\SOFTWARE\Classes\CLSID This can also be done using Regsvr32.exe FakeAMSI.dll. (unregister using Regsvr32.exe /u FakeAMSI.dll)

Next, register the COM server as an AMSI provider by adding the CLSID as a key to HKLM\SOFTWARE \Microsoft\AMSI\Providers.

B.4.3 Requirements

Administrator privileges to create the COM server. It is also required to drop a file.

B.4.4 The achieved persistence

Runs with user privileges. Runs when the AMSI operates. E.g. when opening PowerShell.

B.4.5 More information

- `https://persistence-info.github.io/Data/amsi.html`
- `https://learn.microsoft.com/en-us/windows/win32/amsi/dev-audience`
- `https://github.com/gtworek/PSBits/blob/master/FakeAMSI/FakeAMSI.c`

B.5 AppCert DLL

B.5.1 Origin of the technique

The AppCert DLL is loaded by processes that use one of the following API calls: CreateProcess, CreateProcessAsUser , CreateProcessWithLoginW, CreateProcessWithTokenW, or WinExec. The original functionality of the AppCert DLL is to evaluate the security level of processes started with any of these functions.

B.5.2 How to achieve persistence

Place a DLL someplace and add the path to HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\

B.5.3 Requirements

Administrator privileges.

B.5.4 The achieved persistence

The resulting persistence has user privileges. Trigger is for example by running a command from PowerShell.

B.5.5 More information

- `https://attack.mitre.org/techniques/T1546/009/`
- `https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1546.009/T1546.009.md`
- `https://skanthak.homepage.t-online.de/appcert.html`

B.6 AppInit DLLs

B.6.1 Origin of the technique

Windows has a feature to load default DLLs at the start of any program. This is disabled by default, but it can be enabled to load malicious DLLs.

B.6.2 How to achieve persistence

Under the key HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows or HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows, change the values:

- LoadAppInit_DLLs to 1 (REG_DWORD)
- AppInit_DLLs to the DLL you want to load (REG_SZ)

B.6.3 Requirements

System privileges to write to the registry.

B.6.4 The achieved persistence

It launches when any other application is started, with user privileges.

B.6.5 More information

- `https://cocomelonc.github.io/tutorial/2022/05/16/malware-pers-5.html`
- `https://docs.microsoft.com/en-US/windows/win32/dlls/secure-boot-and-appinit-dlls`
- `https://attack.mitre.org/techniques/T1546/010/`

B.7 Application shimming

B.7.1 Origin of the technique

Application shims are used to ensure backward compatibility of programs with the operating system.

B.7.2 How to achieve persistence

Under HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom\, create a key for the targeted application, e.g. firefox .exe. Under that, create a value {<GUID>}.sdb of type QWORD. Next, create the key {<GUID>} under HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion \AppCompatFlags\InstalledSDB\. Set the value of DatabasePath to the location of the shimming database file (.sdb). Also set the DatabaseRuntimePlatform value to a DWORD matching the type of shimming you want to use. Lastly, set DatabaseType to the type of database you are using.

Another option is making changes to existing sdb files.

B.7.3 Requirements

Administrator privileges to write to the registry.

B.7.4 The achieved persistence

How exactly the persistence works differs per type of shimming, but in general the code is executed when the targeted application is ran.

B.7.5 More information

- `https://attack.mitre.org/techniques/T1546/011/`
- `https://www.ired.team/offensive-security/persistence/t1138-application-shimming`
- `https://github.com/SigmaHQ/sigma/blob/33b370d49bd6aed85bd23827aa16a50bd06d691a/rules/windows/registry/registry_set/registry_set_shim_databases_persistence.yml`
- `https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html`

B.8 App paths

B.8.1 Origin of the technique

Windows stores some of the locations of command in the registry. More specifically, when a program uses the ShellExecuteEx system call and only provides the name of the program, Windows will look in the registry for the location of the program. An example is the run prompt. This persistence techniques changes the value of this registry key to run malicious code instead.

B.8.2 How to achieve persistence

Add a command in HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths or HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths and set the (Default) value to the location of the executable.

B.8.3 Requirements

Administrator privileges to write to the registry, if you write to HKLM. Otherwise user privileges suffice.

B.8.4 The achieved persistence

When a program uses ShellExecuteEx with only the name of a program, the program specified in the registry will be executed.

B.8.5 More information

- `https://www.hexacorn.com/blog/2013/01/19/beyond-good-ol-run-key-part-3/`
- `https://www.cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist`

B.9 APPX

B.9.1 Origin of the technique

Universal Windows Platform apps are a kind of Windows application, distributed using the APPX file format. These include for example the Cortana app and People app. Universal Windows Platform apps allow to automatically run a debugger when the app is started.

B.9.2 How to achieve persistence

Add a registry key with a command you want to execute. Registry keys:

- HKCU\Software\Microsoft\Windows\CurrentVersion\PackagedAppXDebug\<some appx app>\
- HKCU\Software\Classes\ActivatableClasses\Package\<some appx app>\DebugInformation\<a specific version based string>.mca -¿ DebugPath value

B.9.3 Requirements

User privileges to write to the registry.

B.9.4 The achieved persistence

User privileges, when the targeted app is started. Some of them might already start at boot.

B.9.5 More information

- `https://oddvar.moe/2018/09/06/persistence-using-universal-windows-platform-apps-appx/`

B.10 Autodial DLL

B.10.1 Origin of the technique

When using the Winsock library is used to connect to a remote host, the process loads the AutodialDLL. The location of the DLL is set in the registry and can be hijacked.

B.10.2 How to achieve persistence

Replace the registry value AutodialDLL under HKLM\SYSTEM\CurrentControlSet\Services\WinSock2\Parameters with your own DLL.

B.10.3 Requirements

System privileges to write to the registry key. The DLL has to export 3 functions:

- WSAttemptAutodialAddr
- WSAttemptAutodialName
- WSNoteSuccessfulHostentLookup

B.10.4 The achieved persistence

User privileges, every time some internet connection made.

B.10.5 More information

- `https://persistence-info.github.io/Data/autodialdll.html`
- `https://www.hexacorn.com/blog/2015/01/13/beyond-good-ol-run-key-part-24/`
- `https://learn.microsoft.com/en-us/windows/win32/rras/autodial-connection-operations`

B.11 Autoplay handlers

B.11.1 Origin of the technique

When a new device is connected to the computer, a Device Event Handler is used to specify the software to be used to open the medium.

B.11.2 How to achieve persistence

What event (such as a movie dvd is inserted) is matched to event handler. This is written in the registry: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\AutoplayHandlers \EventHandlers They match to names in thehandlers registry. You can change a Autoplay handler in the registry: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\explorer\AutoplayHandlers \Handlers\ Some interesting values are:

- Action - specify exe
- InitCmdLine - specify arguments

B.11.3 Requirements

Admin privileges

B.11.4 The achieved persistence

User privileges, when an attached medium is opened by autoplay. Can run an exe or a DLL.

B.11.5 More information

- `https://www.hexacorn.com/blog/2019/09/07/beyond-good-ol-run-key-part-114/`
- `https://learn.microsoft.com/en-us/windows/win32/shell/how-to-register-a-handler-for-a-device-event`

B.12 BITS jobs

B.12.1 Origin of the technique

Microsoft Windows Background Intelligent Transfer Service (BITS) is used for file transfer between machines and is accessible as a COM server. It is used for example by Windows Update to download updates. The option /SetNotifyCmdLine is used to specify a command that is executed when a job is done transferring or in error.

B.12.2 How to achieve persistence

Set up a BITS job using the following commands:

```
bitsadmin /create my_persistence
bitsadmin /addfile my_persistence c:\windows\system32\net.exe c:\users\ieuser\
    desktop\hi.log
bitsadmin /SetNotifyCmdLine my_persistence "c:\path\to\executable" NULL
bitsadmin /Resume my_persistence
```

BITS jobs are not stored in the registry but in a separate database.

B.12.3 Requirements

User privileges are enough. The database storing the jobs is in %ALLUSERSPROFILE%\Microsoft\Network \Downloader\qmgr.db using the Extensible Storage Engine (ESE) format. "QMGR database files are opened without sharing by the BITS service thus preventing other programs from directly opening them.". In other words, you have to use the bitsadmin tool or the COM object.

Before windows 10 it was stored in qmgr0.dat and qmgr1.dat

B.12.4 The achieved persistence

The command is executed periodically. It stays active for 90 days, or after download is completed.

B.12.5 More information

- `https://www.mandiant.com/resources/blog/attacker-use-of-windows-background-intelligent-transfer-service`
- `https://www.trustedsec.com/blog/bits-persistence-for-script-kiddies/`
- `https://attack.mitre.org/techniques/T1197/`
- `https://unit42.paloaltonetworks.com/unit42-uboatrat-navigates-east-asia/`
- `https://www.secureworks.com/blog/malware-lingers-with-bits`

B.13 Boot verification program

B.13.1 Origin of the technique

After a successful boot, Winlogon notifies the Service Control Manager (SCM) that the boot was successful. To define a successful boot differently, you can add your own program.

B.13.2 How to achieve persistence

Create the registry key HKLM\System\CurrentControlSet\Control\BootVerificationProgram and set ImagePath to your own executable.

B.13.3 Requirements

Administrator privileges to write to the registry.

B.13.4 The achieved persistence

System privileges and runs on every boot.

B.13.5 More information

- `https://www.cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist`

B.14 Browser extensions

B.14.1 Origin of the technique

Browsers have extension that are active when the browser is running.

B.14.2 How to achieve persistence

Add a extension to the default location.

- Chrome:

C:\Users\%username%\AppData\Local\Google\Chrome\User Data\Default\Extensions

- Edge: C:\Users\%username%\AppData\Local\Microsoft\Edge\User Data\Default\Extensions\
- Firefox: C:\Users\%username%\AppData\Roaming\Mozilla\Firefox\Profiles\YOUR_PROFILE.default \extensions\

You can also change the link file on the Desktop to add a custom extension folder (see `https://securityintelligence.com/posts/grandoreiro-malware-now-targeting-banks-in-spain/)`

B.14.3 Requirements

User privileges. Modern browser require confirmation by the user to add an extention.

B.14.4 The achieved persistence

You can run some actions, but only within the browser context.

B.14.5 More information

- `https://mattfrisbie.substack.com/p/spy-chrome-extension`
- `https://attack.mitre.org/techniques/T1176/`

B.15 Change default file association

B.15.1 Origin of the technique

This technique changes the executable that files are opened with. For example, you can specify to open files ending with .txt with a custom application.

B.15.2 How to achieve persistence

Change the registry key HKEY_CLASSES_ROOT\<extension>file\shell\open\command Also related, but for when printing file:

- HKEY_CLASSES_ROOT\txtfile\shell\print\command
- HKEY_CLASSES_ROOT\txtfile\shell\printto\command

B.15.3 Requirements

Admin privileges to write to the registry.

B.15.4 The achieved persistence

When you open a file with the file extension targeted, it will run your executable (with the specified arguments) instead. The privileges depend on if the user opens the file as administrator or not.

B.15.5 More information

- `https://attack.mitre.org/techniques/T1546/001/`
- `http://woshub.com/managing-default-file-associations-in-windows-10/`
- `https://news.sophos.com/en-us/2022/02/01/solarmarker-campaign-used-novel-registry-changes-to-establish-persistence/`

B.16 chm helper dll

B.16.1 Origin of the technique

CHM files are compiled HTML help files. When opened, the chm helper DLL is loaded

B.16.2 How to achieve persistence

Place a malicious DLL and add it to the registry: HKCU\Software\Microsoft\HtmlHelp Author, value location.

B.16.3 Requirements

User privileges.

B.16.4 The achieved persistence

The user has to open a chm file. User privileges.

B.16.5 More information

- `https://persistence-info.github.io/Data/htmlhelpauthor.html`
- `https://www.hexacorn.com/blog/2018/04/22/beyond-good-ol-run-key-part-76/`

B.17 cmd AutoRun

B.17.1 Origin of the technique

The Windows command prompt (cmd) enables the user to set an autorun command that is executed when cmd is started.

B.17.2 How to achieve persistence

Set the value of Autorun of the registry key HKCU\Software\Microsoft\Command Processor\. You can also set it for all users under HKLM\Software\Microsoft\Command Processor.

B.17.3 Requirements

User privileges suffice. Admin privileges required for the system-wide registry.

B.17.4 The achieved persistence

When cmd is run, the command is executed first. The privileges depend on with what privileges the user executes the command prompt.

B.17.5 More information

- `https://persistence-info.github.io/Data/cmdautorun.html`

B.18 Code signing DLL

B.18.1 Origin of the technique

Windows has some code signing DLLs that are used to verify the security of files.

B.18.2 How to achieve persistence

You can register a new code signing DLL. This can be done by adding a key under HKLM\SOFTWARE \Microsoft\Cryptography\OID\EncodingType 0\<some function>\<CLSID>. There are multiple options for functions you can target, see the links below. Set the value DLL to the location of your DLL and FuncName to the name of the targeted function. The placed DLL should export the targeted function.

B.18.3 Requirements

Administrator privileges to write to the registry.

B.18.4 The achieved persistence

The code is run when the DLL is used, for example when displaying file properties.

B.18.5 More information

- `https://persistence-info.github.io/Data/codesigning.html`
- `https://github.com/SigmaHQ/sigma/blob/16990093933fe8c82c3ad879d306d706b350162a/rules/windows/registry/registry_set/registry_set_sip_persistence.yml`
- `https://specterops.io/wp-content/uploads/sites/3/2022/06/SpecterOps_Subverting_Trust_in_Windows.pdf`
- `https://github.com/gtworek/PSBits/tree/master/SIP`

B.19 COM hijack

B.19.1 Origin of the technique

The Component Object Model (COM) enables inter-process communication object creation. When the attackers COM is loaded instead of the intended COM, this is called a COM hijack.

B.19.2 How to achieve persistence

There are multiple approaches to COM hijack, including:

- By CLSID - create a new entry under HKEY_CURRENT_USER\Software\Classes\CLSID matching the CLSID a program is looking for. Since the search order (for non-elevated processes) is to first look in the HKCU registry, this can hijack the COM object.
- By ProgID - some processes to the ProgID to find a COM object instead of the CLSID. The CLSID is then stored inHKEY_CURRENT_USER\Software\Classes\<ProgID>\CLSID
- By TreatAs - can redirect to another COM.

But the main pointer to the exe or DLL is stored in LocalServer32 or InprocServer32 respectively, under HKEY_CURRENT_USER\Software\Classes\CLSID\{GUUID}

B.19.3 Requirements

User privileges for non-elevated COM objects, otherwise higher privleges.

B.19.4 The achieved persistence

Runs when the COM object is loaded by another program. Non-destructive because you can use a COM-Proxy (e.g. `https://github.com/leoloobeek/COMProxy`). Can run both exe (in LocalServer32) or DLL (in InprocServer32).

B.19.5 More information

- `https://cocomelonc.github.io/tutorial/2022/05/02/malware-pers-3.html`
- `https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-2-com-hijacking/`
- `https://attack.mitre.org/techniques/T1546/015/`
- `https://stmxcsr.com/persistence/com-hijacking.html`
- `https://github.com/SigmaHQ/sigma/blob/16990093933fe8c82c3ad879d306d706b350162a/rules/windows/registry/registry_set/registry_set_persistence_search_order.yml`

B.20 Context Menu handler

B.20.1 Origin of the technique

When you right-click a file in Windows explorer or on the Desktop, a menu shows up. It is possible to extend the features of this menu by registering a DLL.

B.20.2 How to achieve persistence

You can add Context menu handler DLLs in multiple locations in the registry, including:

- HKCU\Software\Classes\*\ShellEx\ContextMenuHandlers
- HKCU\Software\Classes\Directory\ShellEx\ContextMenuHandlers
- HKCU\Software\Classes\Directory\Background\ShellEx\ContextMenuHandlers
- HKLM\Software\Classes\*\ShellEx\ContextMenuHandlers
- HKLM\Software\Classes\Directory\ShellEx\ContextMenuHandlers
- HKLM\Software\Classes\Directory\Background\ShellEx\ContextMenuHandlers

Under this key, add a key with as (Default) value a valid CLSID of a COM DLL. The registered DLL should be a valid Context menu handler DLL.

B.20.3 Requirements

User privileges are required.

B.20.4 The achieved persistence

When the user right-click a file, the process is launched, with user privileges.

B.20.5 More information

- `https://pentestlab.blog/2023/03/13/persistence-context-menu/`
- `https://ristbs.github.io/2023/02/15/hijack-explorer-context-menu-for-persistence-and-fun.html`
- `https://github.com/RistBS/ContextMenuHijack`

B.21 COR_PROFILER

B.21.1 Origin of the technique

The .NET framework allows developers to measure their code performance with a user-provided DLL.

B.21.2 How to achieve persistence

You have to set a few environment variables:

- COR_ENABLE_PROFILING=1
- COR_PROFILER={<CLSID of profiler>}
- COR_PROFILER_PATH=<full path of the profiler DLL>

You can store environment variables in the registry under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet \Control\Session Manager\Environment or HKEY_CURRENT_USER\Environment

B.21.3 Requirements

User privileges are enough to set it for the current user.

B.21.4 The achieved persistence

Each .NET process that loads the Common Language Runtime (CLR) will load the malicious DLL. For example, when starting PowerShell. It requires reboot to work.

B.21.5 More information

- `https://redcanary.com/blog/cor_profiler-for-persistence/`
- `https://attack.mitre.org/techniques/T1574/012/`

B.22 Credential Manager DLL

B.22.1 Origin of the technique

When logging in, Winlogon launches mpnotify, which loads various credential manager DLLs. We can add a malicious DLL to this list.

B.22.2 How to achieve persistence

Place the malicious DLL in the System32 folder. Add the name of the DLL to the end of ProviderOrder in HKLM\SYSTEM\CurrentControlSet\Control\NetworkProvider\Order. Finally, create HKLM\SYSTEM \CurrentControlSet\Services\<name>\NetworkProvider and set:

- "Class" = 2 (REG_DWORD)
- "ProviderPath" = "%SystemRoot%\System32\<name>.dll" (REG_EXPAND_SZ)
- "Name" = "<name>" (REG_SZ)

B.22.3 Requirements

System privileges to write the file and write to the registry.

B.22.4 The achieved persistence

Runs with System privileges when the user logs in.

B.22.5 More information

- `https://persistence-info.github.io/Data/credmandll.html`
- `https://github.com/gtworek/PSBits/tree/master/PasswordStealing/NPPSpy`

B.23 Disk Cleanup Handler

B.23.1 Origin of the technique

The Disk Cleanup tool lets the user automatically delete unneeded files. This tool loads DLLs and you can add a DLL to this list.

B.23.2 How to achieve persistence

Set (Default) under HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\VolumeCaches\MyPersistence to {<malicious_CLSID>} Then, hijack the COM DLL (see COM hijack) matching the CLSID. E.g. HKCR \CLSID\<BADCLSID>\InProcServer32 = <dll path>

B.23.3 Requirements

Admin privileges.

B.23.4 The achieved persistence

When the Disk Cleanup tool is launched, the DLL is loaded.

B.23.5 More information

- `https://persistence-info.github.io/Data/diskcleanuphandler.html`
- `https://www.hexacorn.com/blog/2018/09/02/beyond-good-ol-run-key-part-86/`
- `https://learn.microsoft.com/en-us/windows/win32/lwef/disk-cleanup`

B.24 DLL hijack

B.24.1 Origin of the technique

Windows programs can use Dynamic-Link Libraries (DLLs) to leverage functions that are used by many programs. If the DLL used by an executable that is ran by the system can be changed to the DLL provided by the attacker, this is called a DLL hijack.

B.24.2 How to achieve persistence

There are many flavours of DLL hijacks, including:

- DLL replacement (simply replace the DLL file)
- DLL search order hijack (if the full path is not specified)
- Phantom DLL hijack (when a non-existing DLL is loaded)

But there are many more.

B.24.3 Requirements

All techniques involve writing to a DLL file.

B.24.4 The achieved persistence

Runs when the DLL is loaded by an application. Non-destructive, because one can still perform the intended funcions by DLL proxying.

B.24.5 More information

- `https://www.wietzebeukema.nl/blog/hijacking-dlls-in-windows`

B.25 ErrorHandler cmd Hijack

B.25.1 Origin of the technique

Some executables under C:\WINDOWS\system32\oobe\ use a custom error handler command when they experience an error.

B.25.2 How to achieve persistence

Write a command to c:\WINDOWS\Setup\Scripts\ErrorHandler.cmd

B.25.3 Requirements

Admin privileges to write to the file.

B.25.4 The achieved persistence

When some programs in the oobe directory crash (e.g. run setup.exe w/o any arguments). The command will run with user privileges.

B.25.5 More information

- `https://www.hexacorn.com/blog/2022/01/16/beyond-good-ol-run-key-part-135/`

B.26 Explorer tools

B.26.1 Origin of the technique

Windows has various "explorer" tools that are launched automatically or by some user actions. Some example are the Cleanup tool and the Backup tool.

B.26.2 How to achieve persistence

By replacing the (Default) value of one or multiple of the explorer tools in the registry (under HKLM \SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\MyComputer), you can start your own executables instead. The tools we were able to replace are "cleanup", "backup", and "defragmentation". You can also give parameters to the executable.

B.26.3 Requirements

Administrator rights are required.

B.26.4 The achieved persistence

Sometimes executes automatically, otherwise for example when clicking disk cleanup (Under C drive settings).

B.26.5 More information

- `https://persistence-info.github.io/Data/explorertools.html`
- `https://www.hexacorn.com/blog/2017/01/18/beyond-good-ol-run-key-part-55/`
- `https://github.com/SigmaHQ/sigma/blob/1a57509e858674a5acf297061d64b3ffe59e6b3d/rules/windows/registry/registry_set/registry_set_persistence_mycomputer.yml`

B.27 Filter handlers

B.27.1 Origin of the technique

Windows allows for custom filters (ifilters) for files with custom file extensions. When searching in the Windows menu, these filters are applied.

B.27.2 How to achieve persistence

Register a COM DLL for a file extension. Then, add the CLSID under (Default) in HKLM\Software\Classes \.<some extension>\Handler

B.27.3 Requirements

Administrator privileges are required to write to the registry.

B.27.4 The achieved persistence

When a new file with the extension is created, or when the file shows up in a search, the DLL is executed with System privileges.

B.27.5 More information

- `https://persistence-info.github.io/Data/ifilters.html`
- `https://twitter.com/0gtweet/status/1468548924600459267`

B.28 Get-Variable

B.28.1 Origin of the technique

PowerShell loads the Get-Variable cmdlet when it is started.

B.28.2 How to achieve persistence

Drop %APPDATA%\Local\Microsoft\WindowsApps\Get−Variable.exe so that PowerShell will load this cmdlet instead.

B.28.3 Requirements

User privileges are enough to place the file.

B.28.4 The achieved persistence

When the user opens PowerShell the code is executed.

B.28.5 More information

- `https://blog.malwarebytes.com/threat-intelligence/2022/04/colibri-loader-combines-task-scheduler-and-powershell-in-clever-persistence-technique/`

B.29 Group policy logon script

B.29.1 Origin of the technique

You can add logon startup/logon scripts to the group policy.

B.29.2 How to achieve persistence

The registry paths are

- HKCU\Software\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\Logon\<id>\<id>
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\Scripts\<id>\<id>
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\State\Machine\Scripts\<id>\< id>

At the registry key, write:

- Script: the executable
- Parameters: arguments for the executable

The added items are not visible in the UI.

B.29.3 Requirements

Admin privileges to write to the registry.

B.29.4 The achieved persistence

Runs on startup/logon as system.

B.29.5 More information

We did not find earlier descriptions of this technique.

B.30 hhctrl

B.30.1 Origin of the technique

DLL hijack for opening chm files.

B.30.2 How to achieve persistence

Replace CLSID entry HKLM\SOFTWARE\Classes\CLSID\{52A2AAAE–085D–4187–97EA–8C30DB990436 }\InprocServer32 with your own DLL.

B.30.3 Requirements

TrustedInstaller privileges.

B.30.4 The achieved persistence

When a .chm file is openend, you get a user privileges process.

B.30.5 More information

- `https://www.hexacorn.com/blog/2018/04/23/beyond-good-ol-run-key-part-77/`

B.31 Image file execution options

B.31.1 Origin of the technique

Windows allows for starting a debugger instead of the original exe file. The idea is that the debugger can then launch the original exe file as child.

B.31.2 How to achieve persistence

Add HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\{name of the executable} And add value: ”Debugger”=”{full path to the debugger}”

B.31.3 Requirements

Administrator privileges are required to write to the registry.

B.31.4 The achieved persistence

When the targeted executable is started, the defined 'debugger' is started instead. The same privileges that the original executable would get are given to the 'debugger'.

B.31.5 More information

- `https://persistence-info.github.io/Data/ifeo.html`
- `https://blog.malwarebytes.com/101/2015/12/an-introduction-to-image-file-execution-options/`
- `https://oddvar.moe/2018/04/10/persistence-using-globalflags-in-image-file-execution-options-hidden-from-autoruns-exe/`
- `https://attack.mitre.org/techniques/T1546/012/`

B.32 iphlpapi DLL

B.32.1 Origin of the technique

OneDrive and Teams load the iphlpapi.dll DLL if you place it next to them (i.e. in AppData\Local\Microsoft\<OneDrive/Teams>)

B.32.2 How to achieve persistence

Place iphlpapi.dll next to them (i.e. in AppData\Local\Microsoft\<OneDrive/Teams>)

B.32.3 Requirements

User privileges to write the file.

B.32.4 The achieved persistence

Runs when OneDrive or Teams is started.

B.32.5 More information

- `https://blog.cyble.com/2022/07/27/targeted-attacks-being-carried-out-via-dll-sideloading/`
- `https://github.com/SigmaHQ/sigma/blob/bc2188c72bc89469bf425ee3b1a174d58f5586ed/rules/windows/file_event/file_event_win_iphlpapi_dll_sideloading.yml`

B.33 KeePass plugin

B.33.1 Origin of the technique

Keepass is a password manager and allows the installation of 3rd party plugins.

B.33.2 How to achieve persistence

Place a custom DLL in the plugin folder (C:\Program Files (x86)\KeePass Password Safe 2\Plugins)

B.33.3 Requirements

Keepass needs to be installed and system privileges are required.

B.33.4 The achieved persistence

User privileges, runs when Keepass starts.

B.33.5 More information

- `https://medium.com/@two06/persistence-with-keepass-part-1-d2e705326aa6`

B.34 KeePass triggers

B.34.1 Origin of the technique

Keepass allow automation for the user through 'triggers'.

B.34.2 How to achieve persistence

Add a trigger that executes a command by writing to the Keepass config file. (%appdata%\KeePass\KeePass .config.xml)

B.34.3 Requirements

User privileges are enough. Keepass should be closed when you write to the file.

B.34.4 The achieved persistence

You can link execution to a KeePass event, such as starting up. User privileges.

B.34.5 More information

- `https://medium.com/@two06/persistence-with-keepass-part-2-3e328b24e117`

B.35 lnk shortcuts

B.35.1 Origin of the technique

In windows you can create shortcuts to executables. An example are most icons on the desktop.

B.35.2 How to achieve persistence

Change a link file to either open your own file, or add tags to the execution so it will execute your own code besides the original application.

B.35.3 Requirements

Only user privileges are required.

B.35.4 The achieved persistence

The malicious code is executed when the user opens the link file.

B.35.5 More information

- `https://securityintelligence.com/posts/grandoreiro-malware-now-targeting-banks-in-spain/`
- `https://www.ired.team/offensive-security/persistence/modifying-.lnk-shortcuts`

B.36 LSA Authentication Packages

B.36.1 Origin of the technique

LSA authentication packages are used when the user authenticates itself. It loads DLLs that are specified in the registry, which can be leveraged.

B.36.2 How to achieve persistence

HKLM\SYSTEM\CurrentControlSet\Control\Lsa\ with the key value of "Authentication Packages"=<target dll>. The DLL should be placed in System32.

B.36.3 Requirements

Admin privileges.

B.36.4 The achieved persistence

Runs with user privileges when the user authenticates him or herself.

B.36.5 More information

- `https://persistence-info.github.io/Data/authenticationpackages.html`
- `https://attack.mitre.org/techniques/T1547/002/`

B.37 LSA extension

B.37.1 Origin of the technique

LSA (Local Security Authority extension) authentication packages are used when the user authenticates itself. It loads DLLs that are specified in the registry, which can be leveraged.

B.37.2 How to achieve persistence

Add your own DLL to the list in HKLM\SYSTEM\CurrentControlSet\Control\LsaExtensionConfig\LsaSrv, key Extensions. Place your DLL ins C:\Windows\System32

B.37.3 Requirements

To write to the registry, you need TrustedInstaller privileges.

B.37.4 The achieved persistence

Runs when the user authenticates, e.g. when logging in.

B.37.5 More information

- `https://persistence-info.github.io/Data/lsaaextension.html`

B.38 LSA Notification packages

B.38.1 Origin of the technique

This technique has the same origin as other LSA techniques. This time it is executed when you change your password and at boot.

B.38.2 How to achieve persistence

Write to the key Notification packages in HKLM\SYSTEM\CurrentControlSet\Control\Lsa. Write your DLL to System32.

B.38.3 Requirements

Admin privileges.

B.38.4 The achieved persistence

Executes when you change your password, but also on boot. System privileges.

B.38.5 More information

- `https://persistence-info.github.io/Data/passwordfilter.html`

B.39 LSA Security packages

B.39.1 Origin of the technique

Same origin as the other LSA techniques.

B.39.2 How to achieve persistence

Write DLL name to HKLM\SYSTEM\CurrentControlSet\Control\Lsa\, key Security Packages or HKLM \SYSTEM\CurrentControlSet\Control\Lsa\OSConfig\, key Security Packages and place the DLL in system32

B.39.3 Requirements

Admin priviliges.

B.39.4 The achieved persistence

When the user authenticates himself.

B.39.5 More information

- `https://attack.mitre.org/techniques/T1547/005/`

B.40 MPNotify winlogon

B.40.1 Origin of the technique

Winlogon loads various executables and DLLs from the registry.

B.40.2 How to achieve persistence

Add an execuable to registry key: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon
Options are:

- Notify - points to notification package DLLs that handle Winlogon events
- Userinit - points to userinit.exe, the user initialization program executed when a user logs on.
- Shell - points to explorer.exe, the system shell executed when a user logs on.
- mpnotify - can point to an exe.

B.40.3 Requirements

Administrator privileges to write to the registry.

B.40.4 The achieved persistence

For Userinint, the executable is loaded on logon, and should be executed for 30 seconds.

B.40.5 More information

- `https://persistence-info.github.io/Data/mpnotify.html`
- `https://attack.mitre.org/techniques/T1547/004/`
- `https://web.archive.org/web/20201006201906/https://blogs.blackberry.com/en/2013/09/windows-registry-persistence-part-2-the-run-keys-and-search-order`

B.41 Natural Language DLLs

B.41.1 Origin of the technique

SearchIndexer.exe loads some DLLs specified in the registry.

B.41.2 How to achieve persistence

Under any of the languages in HKLM\System\CurrentControlSet\Control\ContentIndex\Language\<some language> set the value of either StemmerDLLPathOverride or WBDLLPathOverride to the location of your DLL.

B.41.3 Requirements

Admin privileges to write to the registry.

B.41.4 The achieved persistence

SYSTEM privileges at log in.

B.41.5 More information

- `https://persistence-info.github.io/Data/naturallanguage6.html`
- `https://www.hexacorn.com/blog/2018/12/30/beyond-good-ol-run-key-part-98/`

B.42 NET DbgManagedDebugger

B.42.1 Origin of the technique

You can set the Just-In-Time debugger for .NET applications.

B.42.2 How to achieve persistence

Set the DbgManagedDebugger value to your executable plus parameters. The value can be found under:

- HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\.NETFramework
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\

B.42.3 Requirements

Administrator privileges to change the registry.

B.42.4 The achieved persistence

When a .NET application errors or crashes, the executable will be ran.

It did not work on my system

B.42.5 More information

- https://www.hexacorn.com/blog/2013/09/19/beyond-good-ol-run-key-part-4/
- https://learn.microsoft.com/en-us/visualstudio/debugger/debug-using-the-just-in-time-debugger?view=vs-2022

B.43 Netsh Helper DLL

B.43.1 Origin of the technique

Netsh is a Windows tool that is used to perform operations on the network configuration. Netsh loads some DLLs from the registry.

B.43.2 How to achieve persistence

Create a new registry key under HKLM\SOFTWARE\Microsoft\Netsh and make it point to a proper Netsh Helper DLL.

B.43.3 Requirements

Admin privileges to write to the registry.

B.43.4 The achieved persistence

Executed when Netsh is ran. On some systems this happens at boot. User privileges.

B.43.5 More information

- https://pentestlab.blog/2019/10/29/persistence-netsh-helper-dll/
- https://attack.mitre.org/techniques/T1546/007/

B.44 NET startup hooks

B.44.1 Origin of the technique

.NET core applications load a DLL at the start if the DOTNET_STARTUP_HOOKS environment variable is set.

B.44.2 How to achieve persistence

Write the location of the malicious DLL to the Environment DOTNET_STARTUP_HOOKS registry key. You can store environment variables in the registry under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet \Control\Session Manager\Environment or HKEY_CURRENT_USER\Environment.

B.44.3 Requirements

User privileges if you want to set only for the current user. The format of the DLL is special, and has to be written in C#.

B.44.4 The achieved persistence

Some .NET core application has to launch.

B.44.5 More information

- https://persistence-info.github.io/Data/dotnetstartuphooks.html
- https://github.com/dotnet/runtime/blob/main/docs/design/features/host-startup-hook.md

B.45 Path interception

B.45.1 Origin of the technique

When running a command from the command line, it looks up the executable through the order of the PATH environment variable.

B.45.2 How to achieve persistence

Change the Path env variable to include a directory that the you also put an executable in. You can store environment variables in the registry under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet \Control\Session Manager\Environment or HKEY_CURRENT_USER\Environment

B.45.3 Requirements

User privileges to set the registry for the current user.

B.45.4 The achieved persistence

When cmd or powershell starts the targeted executable, it will start your executable instead.

B.45.5 More information

- `https://attack.mitre.org/techniques/T1574/007/`

B.46 PowerShell profile

B.46.1 Origin of the technique

PowerShell has the option to execute a 'profile' script every time it starts.

B.46.2 How to achieve persistence

Place a powershell script in any of the following locations:

- $PSHOME\Profile.ps1
- $PSHOME\Microsoft.PowerShell_profile.ps1
- $Home\[My]Documents\[Windows]PowerShell\Profile.ps1
- $Home\[My]Documents\[Windows]PowerShell\Microsoft.PowerShell_profile.ps1

PSHOME is in C:\Windows\System32\WindowsPowerShell\v1.0\ by default.

B.46.3 Requirements

For the user directory, no elevated privileges are required.

B.46.4 The achieved persistence

The script is executed when PowerShell starts.

B.46.5 More information

- `https://persistence-info.github.io/Data/powershellprofile.html`
- `https://attack.mitre.org/techniques/T1546/013/`

B.47 Print monitors

B.47.1 Origin of the technique

Print monitors are part of the Winodws operating system and convert print jobs to proper formats. The Windows Print Spooler Service or spoolv.exe allows to add DLLs as monitors.

B.47.2 How to achieve persistence

To do it manually, write to registry. Under HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors, create a new key, with a subkey Driver containing the DLL location.

B.47.3 Requirements

Administrator privileges required. Works with a regular DLL file.

B.47.4 The achieved persistence

The DLL is load on boot with System privileges.

B.47.5 More information

- `https://stmxcsr.com/persistence/print-monitor.html`
- `https://cocomelonc.github.io/tutorial/2022/06/19/malware-pers-8.html`
- `https://docs.microsoft.com/en-us/windows/win32/printdocs/addmonitor`
- `https://attack.mitre.org/techniques/T1547/010/`

B.48 Print processors

B.48.1 Origin of the technique

Some drivers the are used for printing.

B.48.2 How to achieve persistence

Create a subkey under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Print\Environments \%ARCHITECTURE%\Print Processors\, at a subkey called Driver, with the name of the DLL. The architecture is for example Windows x64. Place the DLL in C:\Windows\system32\spool\PRTPROCS\x64

B.48.3 Requirements

Administrator privileges are required to write to the registry and to the file. Generic DLL seems to suffice.

B.48.4 The achieved persistence

Runs at boot, as SYSTEM.

B.48.5 More information

- `https://stmxcsr.com/persistence/print-processor.html`
- `https://attack.mitre.org/techniques/T1547/012/`

B.49 RDP startup program

B.49.1 Origin of the technique

When connecting to a Windows machine using RDP, Windows will launch rdpclip, which is used for copying between the remote and local machine. We can hijack this program.

B.49.2 How to achieve persistence

For the registry key HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server\Wds\rdpwd set StartupPrograms to your executable.

B.49.3 Requirements

Administrator privileges to write to the registry.

B.49.4 The achieved persistence

Runs everytime a RDP connection is made.

B.49.5 More information

- `https://www.cyberark.com/resources/threat-research-blog/persistence-techniques-that-persist`
- `https://www.cyberark.com/resources/threat-research-blog/attacking-rdp-from-inside`

B.50 RunOnceEx depend dll

B.50.1 Origin of the technique

Another registry key that windows provides for running processes at log in.

B.50.2 How to achieve persistence

Add a DLL path to key some_val under HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnceEx \<some val>\Depend.

B.50.3 Requirements

Administrator privileges are required to write to the registry.

B.50.4 The achieved persistence

Runs at the next boot.

B.50.5 More information

- `https://oddvar.moe/2018/03/21/persistence-using-runonceex-hidden-from-autoruns-exe/`

B.51 RunOnceEx dll

B.51.1 Origin of the technique

Another registry key that windows provides for running processes at log in.

B.51.2 How to achieve persistence

First register the DLL (i.e. add to HHLM\SOFTWARE\Classes\CLSID\<UUID>\) Add a DLL path to HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnceEx\<some val>.

B.51.3 Requirements

Administrator privileges are required.

B.51.4 The achieved persistence

Runs at the next boot.

B.51.5 More information

- `https://oddvar.moe/2018/03/21/persistence-using-runonceex-hidden-from-autoruns-exe/`

B.52 RunOnceEx exe

B.52.1 Origin of the technique

Another registry key that windows provides for running processes at log in.

B.52.2 How to achieve persistence

Add a command string to Software\Microsoft\Windows\CurrentVersion\RunOnceEx\<some val>

B.52.3 Requirements

Administrator privileges are required.

B.52.4 The achieved persistence

Runs at the next boot.

B.52.5 More information

- `https://oddvar.moe/2018/03/21/persistence-using-runonceex-hidden-from-autoruns-exe/`

B.53 Run registry keys

B.53.1 Origin of the technique

Windows has some registry that are meant for running software at log in.

B.53.2 How to achieve persistence

Place an executable (+ parameters) in one of the following registry keys:

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

B.53.3 Requirements

Writing to the HKCU keys require user privileges, and the HKLM require administrator privileges.

B.53.4 The achieved persistence

Runs on login (HKLM for all user, HKCU only the current user). Runonce is only executed only the first boot and then removed.

B.53.5 More information

- `https://attack.mitre.org/techniques/T1547/001/`

B.54 Screensaver

B.54.1 Origin of the technique

You can set up a custom screensaver in windows.

B.54.2 How to achieve persistence

Write the following values to HKEY_CURRENT_USER\Control Panel\Desktop\:

- ScreenSaveActive = 1 (enables the screensaver)
- SCRNSAVE.EXE = path to exe
- ScreenSaveTimeOut = 5 (time in seconds that the user is inactive before the screensaver is executed).

B.54.3 Requirements

Only user privileges required.

B.54.4 The achieved persistence

Runs the executable when the user has been inactive for some time. When the user becomes active again, the process is stopped. "If screensavers are disabled by group policy, this method cannot be used for persistence."

B.54.5 More information

- `https://attack.mitre.org/techniques/T1546/002/`

B.55 Server Level Plugin DLL

B.55.1 Origin of the technique

Windows server, when set up as DNS server, has a DLL to handle unresolved URLs.

B.55.2 How to achieve persistence

Set the value of ServerLevelPluginDll under HKLM\SYSTEM\CurrentControlSet\Services\DNS\Parameters to the path of your DLL.

B.55.3 Requirements

This works on Windows server machines. Administrator and DNSAdmin privileges required.

B.55.4 The achieved persistence

Runs at boot when the DLL is loaded.

B.55.5 More information

- `https://persistence-info.github.io/Data/serverlevelplugindll.html`

B.56 SilentProcessExit

B.56.1 Origin of the technique

Windows has an option to monitor a running process.

B.56.2 How to achieve persistence

You can set what process monitor you want to run for a certain program. You have to perform 3 actions: Set GlobalFlag in HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options \<exe name>.exe to 512 (dword) Set ReportingMode under HKLM\SOFTWARE\Microsoft\Windows NT\ CurrentVersion\SilentProcessExit\<exe name>.exe to 1 (dword) Set MonitorProcess under HKLM\SOFTWARE \Microsoft\Windows NT\CurrentVersion\SilentProcessExit\<exe name>.exe to the path of the executable. You can add parameters.

B.56.3 Requirements

Administrator privileges are required to write to the registry.

B.56.4 The achieved persistence

Runs when the targeted executable is launched.

B.56.5 More information

- `https://oddvar.moe/2018/04/10/persistence-using-globalflags-in-image-file-execution-options-hidden-from-autoruns-exe/`
- `https://persistence-info.github.io/Data/silentexitmonitor.html`

B.57 Startup folder add

B.57.1 Origin of the technique

Windows has some folders for which it will execute the contents on log in. You can set the folders in the registry.

B.57.2 How to achieve persistence

Change one of the following registry keys (name Startup) to your own path:

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders

Place an executable in one of these folders.

B.57.3 Requirements

If you only want to execute it for the current user, user privileges are enough to place the file and write to the registry.

B.57.4 The achieved persistence

Runs at log in. Downside is that the original startup folder is not run anymore.

B.57.5 More information

- `https://attack.mitre.org/techniques/T1547/001/`
- `https://persistence-info.github.io/Data/startupfolder.html`

B.58 Startup folder

B.58.1 Origin of the technique

Windows has some folder for which it will execute the contents on log in.

B.58.2 How to achieve persistence

Place an executable in on of these folders:

```
Windows NT 6.0 - 10.0 / All Users
%SystemDrive%\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup

Windows NT 6.0 - 10.0 / Current User
%SystemDrive%\Users\%UserName%\AppData\Roaming\Microsoft\Windows\Start Menu\
    Programs\Startup

Windows NT 5.0 - 5.2
%SystemDrive%\Documents and Settings\All Users\Start Menu\Programs\Startup

Windows NT 3.5 - 4.0
%SystemDrive%\WINNT\Profiles\All Users\Start Menu\Programs\Startup
```

B.58.3 Requirements

If you only want to execute it for the current user, user privileges are enough to place the file.

B.58.4 The achieved persistence

Runs at log in.

B.58.5 More information

- `https://attack.mitre.org/techniques/T1547/001/`
- `https://persistence-info.github.io/Data/startupfolder.html`

B.59 Task scheduler

B.59.1 Origin of the technique

The Windows operating system has a feature that let;s you schedule tasks. We can use this to schedule the execution of arbitrary commands.

B.59.2 How to achieve persistence

Scheduled taks tasks are stored in the registry (see `https://labs.withsecure.com/publications/scheduled-task-tampering`).

- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks\<name>
- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Plain\{GUID}

Note that some task information is also stored in C:\Windows\System32\Tasks, but making changes to these file does not have any effect.

Scheduling a task can also be achieved through a COM object

B.59.3 Requirements

To write to the tasks registry, you need to be System. I.e., administrator is not enough. However, when not writing directly to the registry but instead using the task scheduler, user privileges are enough.

B.59.4 The achieved persistence

A task can be scheduled to be executed at boot, at log in, every time frame and more. The privileges can be set to system.

B.59.5 More information

- `https://docs.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page`
- `https://persistence-info.github.io/Data/taskscheduler.html`
- `https://www.fuzzysecurity.com/tutorials/19.html`
- `https://stmxcsr.com/persistence/scheduled-tasks.html`

B.60 Time Providers

B.60.1 Origin of the technique

Windows uses DLLs for time synchronization. You can add a custom time provider.

B.60.2 How to achieve persistence

Add a new key under HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\W32Time\TimeProviders \. Then set:

- DllName - name or location of the DLL
- Enabled - 1 (DWORD)
- InputProvider - 1 (DWORD)

B.60.3 Requirements

Administrator privileges are required to write to the registry.

B.60.4 The achieved persistence

It runs the DLL when the Windows Time service is started, e.g. at boot.

B.60.5 More information

- `https://attack.mitre.org/techniques/T1547/003/`
- `https://learn.microsoft.com/en-us/windows/win32/sysinfo/time-provider?redirectedfrom=MSDN`
- `https://stmxcsr.com/persistence/time-provider.html`

B.61 TS Initial Program

B.61.1 Origin of the technique

Windows allows for setting a program that will always run when a RDP connection is made.

B.61.2 How to achieve persistence

Set InitialProgram to your executable, and fInheritInitialProgram to 1 (DWORD) in one of the following registry keys:

- HKLM\SOFTWARE\Policies\Microsoft\Windows NT\Terminal Services
- HKCU\SOFTWARE\Policies\Microsoft\Windows NT\Terminal Services
- HKLM\SYSTEM\CurrentControlSet\Control\Terminal Server\WinStations\RDP−Tcp

B.61.3 Requirements

Only user privileges for the HKCU key.

B.61.4 The achieved persistence

Run when a RDP connection is made.

B.61.5 More information

- `https://persistence-info.github.io/Data/tsinitialprogram.html`

B.62 Universal App URI

B.62.1 Origin of the technique

Windows uses URIs to locate among other Universal apps, such as the Narrator Feedback-Hub.

B.62.2 How to achieve persistence

When using a Universal App URI, Windows will look for the matching name under HKCU\Software\Classes \<name (e.g. AppXysdfafajwklf34234lskfnsklf>)\Shell\open\command. From this entry, remove the DelegateExecute, and set (Default) to an executable with parameters. Other options are adding a new entry under classes, or changing the DelegateExecute to a valid CLSID.

B.62.3 Requirements

User privileges.

B.62.4 The achieved persistence

Executes when the URI is resolved and followed. For the example above, it is when the user uses the narrator feedback hub.

B.62.5 More information

- `https://github.com/giuliocomi/backoori/`
- `https://giuliocomi.blogspot.com/2019/10/abusing-windows-10-narrators-feedback.html`
- `https://github.com/SigmaHQ/sigma/blob/b4cb047ae720b37b11f8506de7965dc29d5920be/rules/windows/registry/registry_event/registry_event_narrator_feedback_persistance.yml`

B.63 User Init Mpr Logon Script

B.63.1 Origin of the technique

You can set up a script that will be executed on log in by Windows.

B.63.2 How to achieve persistence

Set the value of UserInitMprLogonScript in you environment variables. You can do it either in HKCU\Environment or in HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Session Manager\Environment for all users.

B.63.3 Requirements

User privileges.

B.63.4 The achieved persistence

Executed the command on login.

B.63.5 More information

- `https://attack.mitre.org/techniques/T1037/001/`

B.64 Windows Error Reporting Debugger

B.64.1 Origin of the technique

Windows Error Report (WER) allows to automatically attach a debugger when a program hangs.

B.64.2 How to achieve persistence

Write the location of the executable to the key Debugger in HKLM\Software\Microsoft\Windows\Windows Error Reporting\Hangs\

B.64.3 Requirements

Administrator privileges are required to write to the registry.

B.64.4 The achieved persistence

Executes when a program hangs, but does not work for me.

B.64.5 More information

- `https://www.hexacorn.com/blog/2019/09/20/beyond-good-ol-run-key-part-116/`

B.65 Windows Load

B.65.1 Origin of the technique

When Explorer starts at boot, it executes a binary that is defined in the registry HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows under the Load key.

B.65.2 How to achieve persistence

Set the Load key to the location of your executable.

B.65.3 Requirements

User privileges.

B.65.4 The achieved persistence

Executes when you open Explorer.

B.65.5 More information

- `https://persistence-info.github.io/Data/windowsload.html`

B.66 Windows services

B.66.1 Origin of the technique

Windows has Services as a feature, but they can also be used to gain persistence.

B.66.2 How to achieve persistence

Windows service settings are stored in the registry under HKLM\System\CurrentControlSet\Services You can set the value of Start to 2 to start on boot. Another option is to add the service to one of the following keys:

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices

B.66.3 Requirements

System privileges are required.

B.66.4 The achieved persistence

When the service is set to automatic, it is started at boot and executed every interval. The spawned process has system rights.

B.66.5 More information

- `https://gtworek.github.io/PSBits/services.html`
- `https://cocomelonc.github.io/tutorial/2022/05/09/malware-pers-4.html`
- `https://attack.mitre.org/techniques/T1543/003/`

B.67 Windows Telemetry

B.67.1 Origin of the technique

Windows has some telemetry functions. These are extendable through the registry.

B.67.2 How to achieve persistence

Add a key to HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags \TelemetryController and set:

- Command to your execuable with arguments
- One or more of Maintenance, Nightly, Oobe to 1 (DWORD)

B.67.3 Requirements

Administrator privileges are required to write to the registry key.

B.67.4 The achieved persistence

Runs at least every 24 hours when nightly is set.

B.67.5 More information

- `https://www.trustedsec.com/blog/abusing-windows-telemetry-for-persistence/`
- `https://persistence-info.github.io/Data/telemetrycontroller.html`

B.68 Windows Terminal Profile

B.68.1 Origin of the technique

Windows terminal allows to set up profiles and to start it self on user login, all through the settings file.

B.68.2 How to achieve persistence

Create a new profile in%LOCALAPPDATA%\Packages\Microsoft.WindowsTerminal_8wekyb3d8bbwe\LocalState \settings.json:

```
{
        "closeOnExit": "graceful",
        "commandline": "C:\my\command.exe -w arguments",
        "guid": "{some-guid}",
        "hidden": true,
        "name": "mypersistence"
}
```

Change the defaultProfile to your guid add "startOnUserLogin": true

B.68.3 Requirements

Windows Terminal needs to be installed. User privileges are enough.

B.68.4 The achieved persistence

Runs your command when the user logs in, or when the terminal is opened. It is destructive, as it does not launch the normal terminal. If you opt to add opening the normal terminal to the commandline, then it will open everytime you log in.

B.68.5 More information

- `https://persistence-info.github.io/Data/windowsterminalprofile.html`

B.69 Winlogon GP client-side extension

B.69.1 Origin of the technique

Group Policy Client Side Extension is a DLL that runs on the client computer.

B.69.2 How to achieve persistence

Set DllName under HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GPExtensions \{GUID}\ to the path of your DLL.

B.69.3 Requirements

Admin privileges are required to write to registry.

B.69.4 The achieved persistence

The DLL is loaded when a policy is applied. Should give sytem.

B.69.5 More information

- https://persistence-info.github.io/Data/gpoextension.html
- https://docs.microsoft.com/en-us/previous-versions/windows/desktop/policy/creating-a-policy-callback-function
- https://www.hexacorn.com/blog/2018/10/11/beyond-good-ol-run-key-part-92/

B.70 WMI Event Subscription

B.70.1 Origin of the technique

WMI stands for Windows Management Instrumentation. You can create a EventFilter which queries for a specific event. When this even occurs, a specific action can be executed. The consumer can be CommandLineEventConsumer, which allows for command execution.

B.70.2 How to achieve persistence

WMI events subsriptions are stored in the WMI repository, which can be found in %windir%\System32\Wbem\Repository. You can write to it, but the format is not very straight forward.

B.70.3 Requirements

You need to have admin privileges. You need to write to the directory mentioned above. However, it is a not-so-easy to understand database, so it is virtually impossible to write to it apart from using the provided WMI API.

B.70.4 The achieved persistence

You can connect execution of any command to many events. The command will run as system.

B.70.5 More information

- https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-3-wmi-event-subscription/
- http://www.fuzzysecurity.com/tutorials/19.html
- https://wumb0.in/scheduling-callbacks-with-wmi-in-cpp.html
- https://attack.mitre.org/techniques/T1546/003/
- https://stmxcsr.com/persistence/wmi-persistence.html
- https://www.eideon.com/2018-03-02-THL03-WMIBackdoors/
- https://www.blackhat.com/docs/us-15/materials/us-15-Graeber-Abusing-Windows-Management-Instrumentation-WMI-To-Build-A-Persistent%20Asynchronous-And-Fileless-Backdoor-wp.pdf